Lending Club Loan Data Analysis

Importing Libraries

In [1]:

```
import numpy as np
import pandas as pd
import re   # Regex
pd.options.mode.chained_assignment = None

# Visualizations
import plotly.plotly as py
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from plotly import tools
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()
from ggplot import *
%matplotlib notebook
%matplotlib inline
import seaborn as sns

#Missing Values
import missingno as msno

# import functools
# from tabulate import tabulate
# from subprocess import check_output

import pandasql as pdsql
from pandasql import sqldf
```

Reading Data

We are going to be combining 2014 and 2015 dataset in one dataframe and call it "custdata"

In [2]:

```
custdata14 = pd.read_csv('C:\LoanStats14.csv',low_memory=False)
```

In [3]:

```
custdata14.shape
```

Out[3]:

```
(235633, 122)
```

In [4]:

```
custdata15 = pd.read_csv('C:\LoanStats15.csv',low_memory=False)
```

In [5]:

```
custdata15.shape
```

Out[5]:

```
(421099, 122)
```

In [6]:

```
custdata = pd.concat([custdata14,custdata15])
```

In [7]:

```
custdata.shape
```

Out[7]:

```
(656732, 122)
```

Quick Data Exploration

In [8]:

```
custdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 656732 entries, 0 to 421098
Columns: 122 entries, id to sec_app_mths_since_last_major_derog
dtypes: float64(97), object(25)
memory usage: 616.3+ MB
```

In [9]:

```
custdata.columns.tolist() #List of features
```

Out[9]:

```
['id',
 'member_id',
 'loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'url',
 'desc',
 'purpose',
 'title',
 'zip_code',
```

```
    'addr_state',
    'dti',
    'delinq_2yrs',
    'earliest_cr_line',
    'inq_last_6mths',
    'mths_since_last_delinq',
    'mths_since_last_record',
    'open_acc',
    'pub_rec',
    'revol_bal',
    'revol_util',
    'total_acc',
    'initial_list_status',
    'out_prncp',
    'out_prncp_inv',
    'total_pymnt',
    'total_pymnt_inv',
    'total_rec_prncp',
    'total_rec_int',
    'total_rec_late_fee',
    'recoveries',
    'collection_recovery_fee',
    'last_pymnt_d',
    'last_pymnt_amnt',
    'next_pymnt_d',
    'last_credit_pull_d',
    'collections_12_mths_ex_med',
    'mths_since_last_major_derog',
    'policy_code',
    'application_type',
    'annual_inc_joint',
    'dti_joint',
    'verification_status_joint',
    'acc_now_delinq',
    'tot_coll_amt',
    'tot_cur_bal',
    'open_acc_6m',
    'open_il_6m',
    'open_il_12m',
    'open_il_24m',
    'mths_since_rcnt_il',
    'total_bal_il',
    'il_util',
    'open_rv_12m',
    'open_rv_24m',
    'max_bal_bc',
    'all_util',
    'total_rev_hi_lim',
    'inq_fi',
    'total_cu_tl',
    'inq_last_12m',
    'acc_open_past_24mths',
    'avg_cur_bal',
    'bc_open_to_buy',
    'bc_util',
    'chargeoff_within_12_mths',
    'delinq_amnt',
    'mo_sin_old_il_acct',
    'mo_sin_old_rev_tl_op',
    'mo_sin_rcnt_rev_tl_op',
```

```
 'mo_sin_rcnt_tl',
 'mort_acc',
 'mths_since_recent_bc',
 'mths_since_recent_bc_dlq',
 'mths_since_recent_inq',
 'mths_since_recent_revol_delinq',
 'num_accts_ever_120_pd',
 'num_actv_bc_tl',
 'num_actv_rev_tl',
 'num_bc_sats',
 'num_bc_tl',
 'num_il_tl',
 'num_op_rev_tl',
 'num_rev_accts',
 'num_rev_tl_bal_gt_0',
 'num_sats',
 'num_tl_120dpd_2m',
 'num_tl_30dpd',
 'num_tl_90g_dpd_24m',
 'num_tl_op_past_12m',
 'pct_tl_nvr_dlq',
 'percent_bc_gt_75',
 'pub_rec_bankruptcies',
 'tax_liens',
 'tot_hi_cred_lim',
 'total_bal_ex_mort',
 'total_bc_limit',
 'total_il_high_credit_limit',
 'revol_bal_joint',
 'sec_app_earliest_cr_line',
 'sec_app_inq_last_6mths',
 'sec_app_mort_acc',
 'sec_app_open_acc',
 'sec_app_revol_util',
 'sec_app_open_il_6m',
 'sec_app_num_rev_accts',
 'sec_app_chargeoff_within_12_mths',
 'sec_app_collections_12_mths_ex_med',
 'sec_app_mths_since_last_major_derog']
```

In [10]:

```
custdata.head()
```

Out[10]:

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment
0	NaN	NaN	10400.0	10400.0	10400.0	36 months	6.99%	321.08
1	NaN	NaN	15000.0	15000.0	15000.0	60 months	12.39%	336.64
2	NaN	NaN	9600.0	9600.0	9600.0	36 months	13.66%	326.53
3	NaN	NaN	7650.0	7650.0	7650.0	36 months	13.66%	260.20
4	NaN	NaN	12800.0	12800.0	12800.0	60	17.14%	319.08

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment
						months		

5 rows × 122 columns

In [11]:

```
custdata.describe()
```

Out[11]:

	member_id	loan_amnt	funded_amnt	funded_amnt_inv	installment	annual_i
count	0.0	656724.000000	656724.000000	656724.000000	656724.000000	6.567240
mean	NaN	15107.485565	15107.485565	15101.824997	442.074381	7.620803
std	NaN	8525.684352	8525.684352	8522.493993	244.920019	6.793073
min	NaN	1000.000000	1000.000000	900.000000	14.010000	0.000000
25%	NaN	8450.000000	8450.000000	8450.000000	264.460000	4.600000
50%	NaN	13750.000000	13750.000000	13750.000000	385.110000	6.500000
75%	NaN	20000.000000	20000.000000	20000.000000	578.790000	9.000000
max	NaN	35000.000000	35000.000000	35000.000000	1445.460000	9.500000

8 rows × 97 columns

Quick inferences based on the above.

1. loan_amnt and funded_amnt are having same values.
2. Maximum loan_amnt is 35000
3. Member_id ,url fields are not populated.
4. Many records have NaN entries.

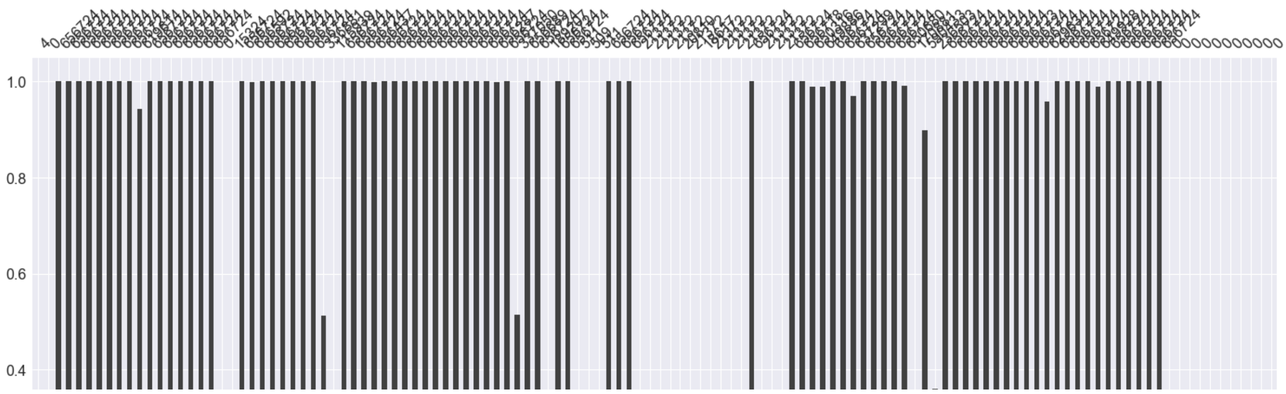
Identifying and visualizing missing values

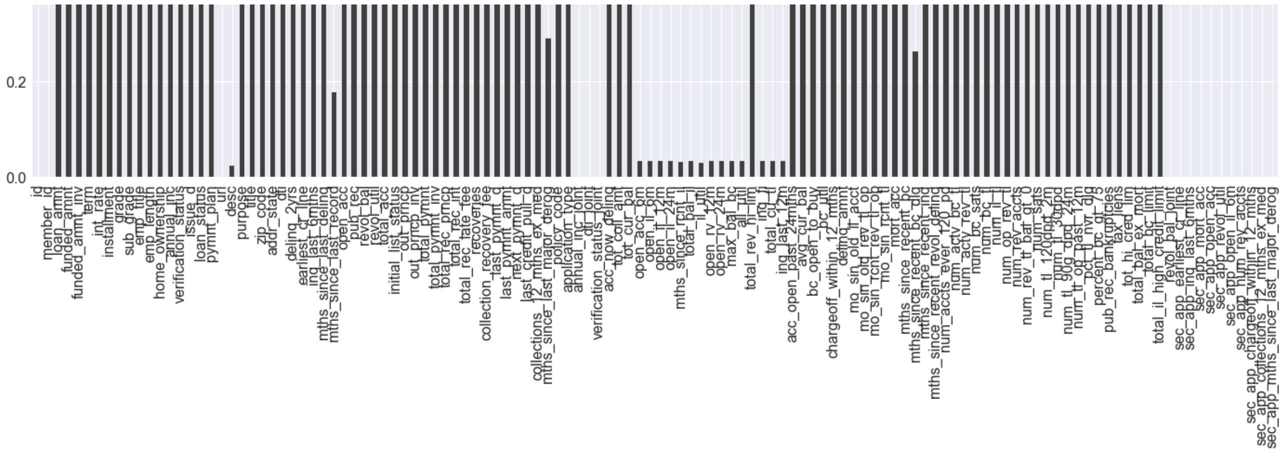
In [12]:

```
# custdata.apply(lambda x: sum(x.isnull()),axis=0)
```

In [13]:

```
msno.bar(custdata)#columns with missing values
```





Above bar chart helps identifying columns with missing values in the dataset. It appears that there are quite a number of columns with missing values. For example:- "url" column is empty. "desc" column has also some missing values. "months since last delinq", "monthsince last record" are not populated. We will be dropping the columns which have missing values upto 75-80%.

Dealing with the columns with missing data

In [14]:

```
custdata = custdata.iloc[:, :-11]#Removed last 11 columns with sparse data
```

In [15]:

```
custdata.shape
```

Out[15]:

```
(656732, 111)
```

Removing columns with nan values

Removed annual_inc_joint,dti_joint,verification_status_joint

In [16]:

```
custdata.drop(['annual_inc_joint','dti_joint','verification_status_joint'],
1, inplace=True)
```

Reviewing columns and identifying features to work with.

We will be focusing on the subset of the data and base our analysis on a selected set of variables. I explored the features in batches, dropped the irrelevant columns which will not be needed in the context of intended analysis. I will be pulling those columns in a separate df 'loandata'.

In [17]:

```
loandata = custdata[['loan_amnt','funded_amnt','int_rate','emp_length','loa
n_status','home_ownership','grade','annual_inc','verification_status','term
','dti','revol_bal','total_acc']]
```

In [18]:

```
loandata.dropna().info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 656724 entries, 0 to 421094
Data columns (total 13 columns):
loan_amnt              656724 non-null float64
funded_amnt            656724 non-null float64
int_rate               656724 non-null object
emp_length             656724 non-null object
loan_status            656724 non-null object
home_ownership         656724 non-null object
grade                  656724 non-null object
annual_inc             656724 non-null float64
verification_status    656724 non-null object
term                   656724 non-null object
dti                    656724 non-null float64
revol_bal              656724 non-null float64
total_acc              656724 non-null float64
dtypes: float64(6), object(7)
memory usage: 70.1+ MB
```

Now we will be pulling the records which have NaN entries. We will drop all those rows which have NaN entries for all the columns.

In [19]:

```
nan_rows = loandata[loandata.isnull().T.any().T]
print(nan_rows)
```

```
        loan_amnt  funded_amnt int_rate emp_length loan_status
home_ownership  \
235629        NaN          NaN      NaN        NaN         NaN              Na

235630        NaN          NaN      NaN        NaN         NaN              Na

235631        NaN          NaN      NaN        NaN         NaN              Na

235632        NaN          NaN      NaN        NaN         NaN              Na

421095        NaN          NaN      NaN        NaN         NaN              Na

421096        NaN          NaN      NaN        NaN         NaN              Na

421097        NaN          NaN      NaN        NaN         NaN              Na

421098        NaN          NaN      NaN        NaN         NaN              Na


       grade  annual_inc verification_status term  dti  revol_bal
total_acc
235629   NaN         NaN                 NaN  NaN  NaN        NaN         NaN

235630   NaN         NaN                 NaN  NaN  NaN        NaN         NaN

235631   NaN         NaN                 NaN  NaN  NaN        NaN         NaN

235632   NaN         NaN                 NaN  NaN  NaN        NaN         NaN

421095   NaN         NaN                 NaN  NaN  NaN        NaN         NaN
```

421096	NaN	NaN	NaN	NaN	NaN	NaN	NaI
421097	NaN	NaN	NaN	NaN	NaN	NaN	NaI
421098	NaN	NaN	NaN	NaN	NaN	NaN	NaI

In [20]:

```
### loandata.info()
loandata = loandata.dropna(how='all')
loandata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 656724 entries, 0 to 421094
Data columns (total 13 columns):
loan_amnt              656724 non-null float64
funded_amnt            656724 non-null float64
int_rate               656724 non-null object
emp_length             656724 non-null object
loan_status            656724 non-null object
home_ownership         656724 non-null object
grade                  656724 non-null object
annual_inc             656724 non-null float64
verification_status    656724 non-null object
term                   656724 non-null object
dti                    656724 non-null float64
revol_bal              656724 non-null float64
total_acc              656724 non-null float64
dtypes: float64(6), object(7)
memory usage: 70.1+ MB
```

In [21]:

```
loandata.apply(lambda x: sum(x.isnull()),axis=0)
```

Out[21]:

```
loan_amnt              0
funded_amnt            0
int_rate               0
emp_length             0
loan_status            0
home_ownership         0
grade                  0
annual_inc             0
verification_status    0
term                   0
dti                    0
revol_bal              0
total_acc              0
dtype: int64
```

Data Munging

Analysing "loan_status"

There are 7 distinct Loan Status categories. Lets look at the distribution:-

Finding no of applicants per loan status

In [22]:

```
loan_status_dist=pd.DataFrame(loandata['loan_status'].value_counts())
loan_status_dist.reset_index(inplace=True)
loan_status_dist.columns=['Status_Category','No. of applicants']
loan_status_dist
```

Out[22]:

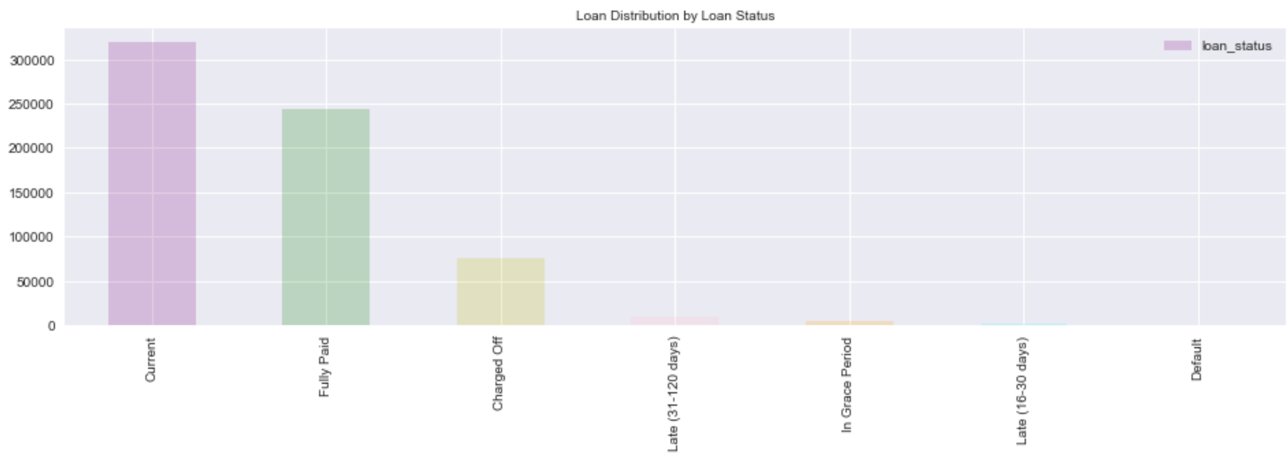
	Status_Category	No. of applicants
0	Current	319220
1	Fully Paid	244085
2	Charged Off	75589
3	Late (31-120 days)	10123
4	In Grace Period	5187
5	Late (16-30 days)	2506
6	Default	14

In [23]:

```
colors = ['purple', 'green', 'y', 'pink','orange','cyan']
loandata.loan_status.value_counts().plot(kind='bar',color=colors,alpha=.20,
legend=True,figsize = [16,4])
plt.title('Loan Distribution by Loan Status', fontsize = 10)
```

Out[23]:

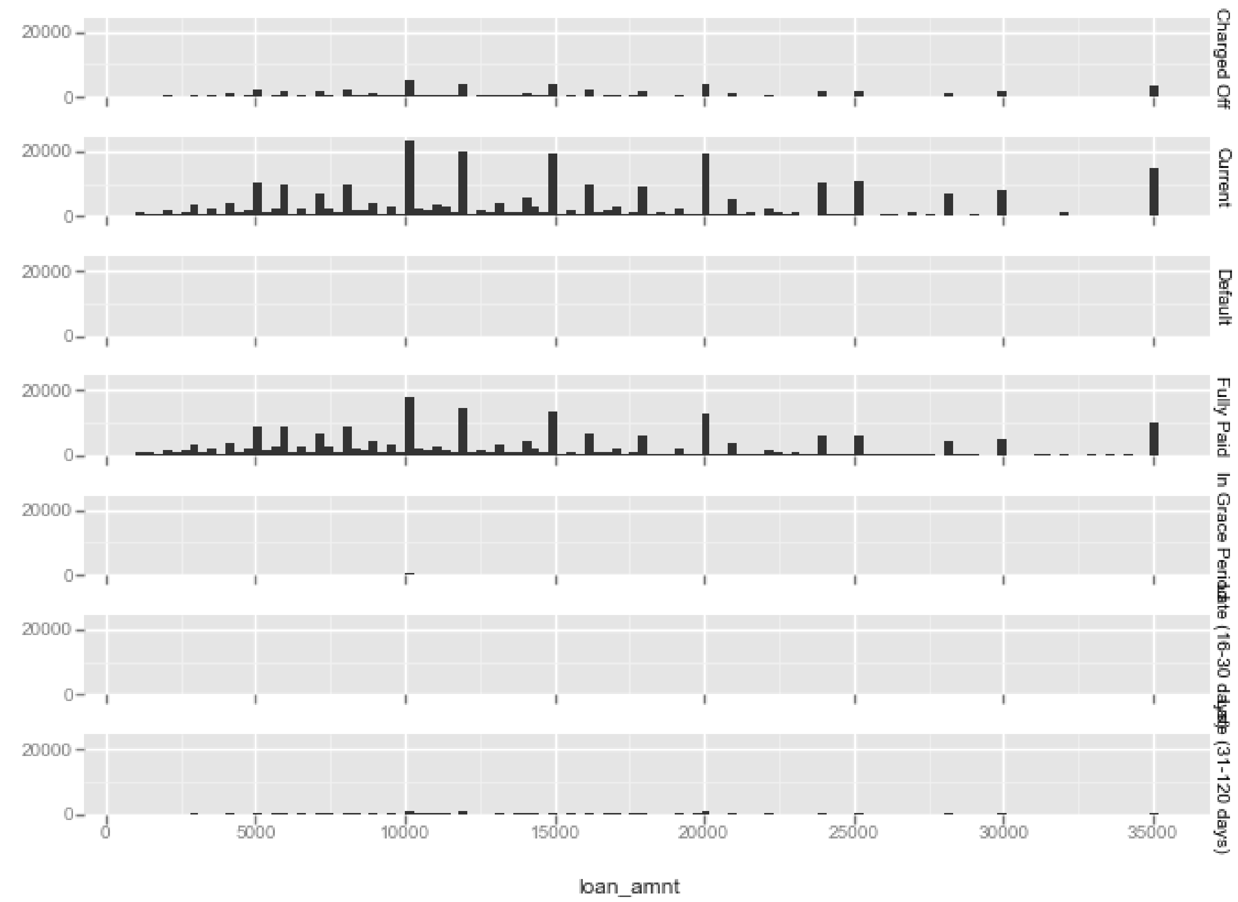
<matplotlib.text.Text at 0x11da77b8>



In [24]:

```
from ggplot import *
# ggplot(loandata, aes(loan_amnt, col = grade)) + geom_histogram(bins =
50) + facet_grid(grade)
gg = ggplot(loandata, aes('loan_amnt',col='loan_status')) + geom_histogram(
binwidth=300)+facet_grid('loan_status')
```

```
print (gg)
```




```
<ggplot: (15114574)>
```

we see a variation of loan amount in the Charged off and Late (31-120 days) status category. We will set the premise of loan prediction on this variable.

Cleaning loan_status

0 = (Charged off, Default) 1 = (Fully -Paid) 2 = (current,late,in grace-period) We will filter all the records with status = current/late/in-grace period. 'late payments' and 'in-grace period' status can turn up or down. After filtering we reduced the records to 319688

In [25]:

```
loandata['loan_status_clean'] = loandata['loan_status'].map({'Current': 2,
'Fully Paid': 1, 'Charged Off':0, 'Late (16-30 days)':2, 'In Grace Period':
2, 'Late (31-120 days)': 2, 'Default': 0})
loandata = loandata[loandata.loan_status_clean != 2]# Getting rid of curren
t loan from the dataset
loandata["loan_status_clean"] = loandata["loan_status_clean"].apply(lambda
loan_status_clean: 0 if loan_status_clean == 0 else 1)
loandata.head(2)
# loandata['loan_new'] = loandata['loan_status'].map({'Current': 'X', 'Full
y Paid': 'Y', 'Charged Off':'N', 'Late(31-120 days)':'X', 'In Grace Period'
: 'X', 'Late(16-30 days)': 'X', 'Default': 'N'})
# loandata = loandata[loandata.loan_new != 'X']
```

Out[25]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu
0	10400.0	10400.0	6.99%	8 years	Charged Off	MORTGAGE	A	5800
1	15000.0	15000.0	12.39%	10+ years	Fully Paid	RENT	C	7800

In [26]:

```
pysql = lambda q: pdsql.sqldf(q, globals())

str1= """SELECT loan_status_clean, loan_status, count(*) as count_of_loan_i
ssued from loandata group by loan_status_clean,loan_status """
df1 = pysql(str1)
# df1.set_index('loan_status',inplace = True)
df1.head(7)
# loandata.head()
```

Out[26]:

	loan_status_clean	loan_status	count_of_loan_issued
0	0	Charged Off	75589
1	0	Default	14
2	1	Fully Paid	244085

In [27]:

```
loandata.loan_status_clean.value_counts()
```

Out[27]:

```
1    244085
0     75603
Name: loan_status_clean, dtype: int64
```

Looks like 'Paid' and 'Charged/Defaulted' loans make up a ratio of approximately 3:1. We will label them 'Good' and 'Bad'. Below bar graph presents the distribution.

In [28]:

```
%matplotlib inline

import matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

In [29]:

```
colors = ['green','orange']
loandata.loan_status_clean.value_counts().plot(kind='bar',color = colors,al
pha=.30,figsize = [16,4])
plt.xlabel("Good Loan = 1 Bad Loan = 0")
plt.ylabel("Cnt")
plt.title("Good Loan vs Bad Loan Count")
# plt.hist(list(fil_loandata['loan_status']))
plt.savefig("2013-2014-broader-bad-loan-def.png")
```

In [30]:

```
loandata.shape
```

Out[30]:

```
(319688, 14)
```

In [31]:

```
loandata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 319688 entries, 0 to 421093
Data columns (total 14 columns):
loan_amnt              319688 non-null float64
funded_amnt            319688 non-null float64
int_rate               319688 non-null object
emp_length             319688 non-null object
loan_status            319688 non-null object
home_ownership         319688 non-null object
grade                  319688 non-null object
annual_inc             319688 non-null float64
verification_status    319688 non-null object
term                   319688 non-null object
dti                    319688 non-null float64
revol_bal              319688 non-null float64
total_acc              319688 non-null float64
loan_status_clean      319688 non-null int64
dtypes: float64(6), int64(1), object(7)
memory usage: 36.6+ MB
```

Analysing "verification status"

verification_status Indicates if income was verified by LC, not verified, or if the income source was verified.

In [32]:

```
loandata.head()
loandata[ (loandata['verification_status'].isnull())]##checking if the colu
mn has null values
```

Out[32]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annua

In [33]:

```
verification_status_dist=pd.DataFrame(loandata['verification_status'].value_counts())
verification_status_dist.reset_index(inplace=True)
verification_status_dist.columns=['Status_Category','No. of applicants']
verification_status_dist
```

Out[33]:

	Status_Category	No. of applicants
0	Source Verified	131978
1	Verified	95833
2	Not Verified	91877

In [34]:

```
import plotly
plotly.offline.init_notebook_mode()
trace=go.Pie(labels=verification_status_dist['Status_Category'],values=verification_status_dist['No. of applicants'])
iplot([trace])
```

Alsmost 30% of the records seem to be 'Not-Verified'. It's appropriate to put' 'Verified' and 'Source Verified' records in one bucket and 'Not-verified' into the other

Verified records in one bucket and Not Verified into the other.

Cleaning Verification Status

Verified or Source Verified= 1 Not Verified = 0

In [35]:

```
loandata['verification_status_clean'] = loandata['verification_status'].map
({'Source Verified': 1, 'Verified': 1, 'Not Verified':0})
loandata["verification_status_clean"] =
loandata["verification_status_clean"].apply(lambda
verification_status_clean: 0 if verification_status_clean == 0 else 1)
loandata.head(2)
```

Out[35]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu
0	10400.0	10400.0	6.99%	8 years	Charged Off	MORTGAGE	A	5800(
1	15000.0	15000.0	12.39%	10+ years	Fully Paid	RENT	C	7800(

In [36]:

```
loandata.verification_status_clean.value_counts()
```

Out[36]:

```
1    227811
0     91877
Name: verification_status_clean, dtype: int64
```

In [37]:

```
# t1 = t/t.ix["Ctotal","Rtotal"]
t1 = pd.crosstab(index=loandata["verification_status_clean"],columns=loanda
ta["loan_status_clean"], margins = True)
t1.columns = ["Bad Loan","Good Loan","Rtotal"]
t1.index = ["Not-Verified","Verified","Ctotal"]
t1
```

Out[37]:

	Bad Loan	Good Loan	Rtotal
Not-Verified	16202	75675	91877
Verified	59401	168410	227811
Ctotal	75603	244085	319688

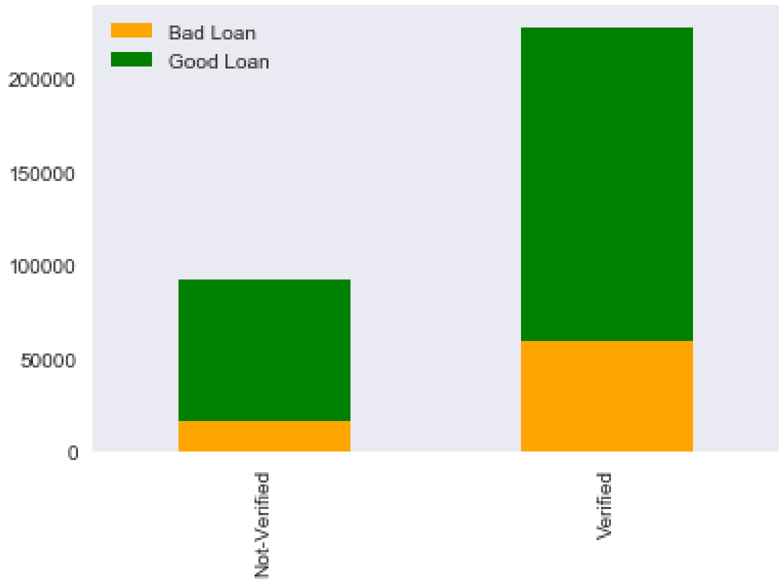
In [38]:

```
t2 = pd.crosstab(index=loandata["verification_status_clean"],columns=loanda
ta["loan_status_clean"])
t2.columns = ["Bad Loan","Good Loan"]
t2.index = ["Not-Verified","Verified"]
```

```
t2.plot(kind='bar', stacked=True, color=['orange','green'], grid=False)
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x113f2da0>
```



Analysing "term"

Term indicates number of payments on the loan. Values are in months and can be either 36 or 60. We will see if term has any impact in a loan getting paid defaulted.

In [39]:

```
loandata[ (loandata['term'].isnull())]##checking if the column has null val
ues
```

Out[39]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annua

In [40]:

```
term_dist=pd.DataFrame(loandata['term'].value_counts())
term_dist.reset_index(inplace=True)
term_dist.columns=['term','No. of applicants']
term_dist
```

Out[40]:

	term	No. of applicants
0	36 months	234926
1	60 months	84762

In [41]:

```
pysql = lambda q: pdsql.sqldf(q, globals())

str1= """SELECT loan_status_clean as defaulted_loan, term, grade,count(*) a
```

```
s count_of_loan_issued
from loandata
group by loan_status_clean,term,grade """
df1 = pysql(str1)
# df1.set_index('loan_status',inplace = True)
df1.head(14)
# loandata.head()
```

Out[41]:

	defaulted_loan	term	grade	count_of_loan_issued
0	0	36 months	A	3471
1	0	36 months	B	10656
2	0	36 months	C	15682
3	0	36 months	D	9773
4	0	36 months	E	3938
5	0	36 months	F	965
6	0	36 months	G	147
7	0	60 months	A	87
8	0	60 months	B	1863
9	0	60 months	C	6811
10	0	60 months	D	8845
11	0	60 months	E	8448
12	0	60 months	F	3719
13	0	60 months	G	1198

Cleaning "term"

36 months = '1', 60 months = '0'

In [42]:

```
loandata['term_clean'] = loandata['term'].map({' 36 months': 1, ' 60
months': 0})
loandata["term_clean"] = loandata["term_clean"].apply(lambda term_clean: 0
if term_clean == 0 else 1)
loandata.head(2)
# loandata.drop(['term_cln'],1, inplace=True)
```

Out[42]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu
0	10400.0	10400.0	6.99%	8 years	Charged Off	MORTGAGE	A	58000
1	15000.0	15000.0	12.39%	10+ years	Fully Paid	RENT	C	78000

In [43]:

```
t1 =
pd.crosstab(index=loandata["term_clean"],columns=loandata["loan_status_clea
"], margins = True)
t1.columns = ["Bad Loan","Good Loan","Rtotal"]
t1.index = ["60 months","36 months","Ctotal"]
t0 = t1/t1.ix["Ctotal","Rtotal"]*100
t0
t1
```

Out[43]:

	Bad Loan	Good Loan	Rtotal
60 months	30971	53791	84762
36 months	44632	190294	234926
Ctotal	75603	244085	319688

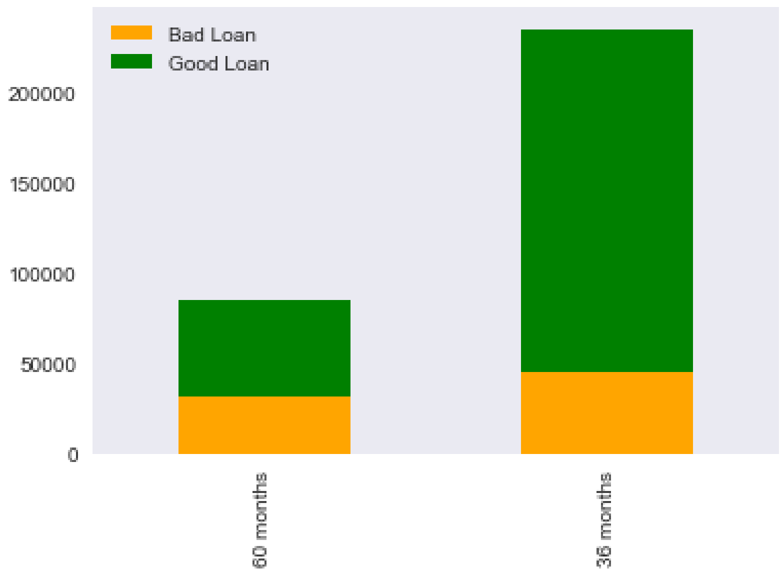
In [44]:

```
t2 = pd.crosstab(loandata['term_clean'], loandata['loan_status_clean'])
t2.columns = ["Bad Loan","Good Loan"]
t2.index = ["60 months","36 months"]
print ("Term vs Loan")

t2.plot(kind='bar', stacked=True, color=['orange','g'], grid=False)
```

```
Term vs Loan
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1184acc0>
```



18 % of the 36 months term loans have turned bad vs 36 % of loan with 60 months term.We will check it's predictive power in the latter section.

Analysing "emp_length"

It indicates employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years. It's intuitive enough to assume that the loan payments would continue as long as individual continues to work but I would like to explore if number of years have anything to do with the loan defaults emphasizing more on the lower range(1-3 years) of emp_length.

In [45]:

```
loandata[ (loandata['emp_length'].isnull())]##checking if the column has nu
ll values
```

Out[45]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annua

In [46]:

```
loandata["emp_length"].unique()
```

Out[46]:

```
array(['8 years', '10+ years', '< 1 year', '6 years', '2 years', '7 years',
       '9 years', 'n/a', '3 years', '1 year', '4 years', '5 years'],
dtype=object)
```

In [47]:

```
###EMP_LENGTH
print(loandata.emp_length.value_counts())
loandata.emp_length.unique().shape
```

```
10+ years    104925
2 years       28829
< 1 year      25992
3 years       25486
1 year        20807
5 years       18608
4 years       18600
n/a           16370
8 years       16365
7 years       16163
6 years       14710
9 years       12833
Name: emp_length, dtype: int64
```

Out[47]:

```
(12,)
```

Lets look at the "n/a" records against good/bad loan counts.

In [48]:

```
pysql = lambda q: pdsql.sqldf(q, globals())

str1= """SELECT emp_length,loan_status_clean,  count(*) as count_of_loan_is
sued
from loandata
where emp_length = 'n/a'
```

```
where emp_length = 'n/a'
group by emp_length,loan_status_clean """
df1 = pysql(str1)
# df1.set_index('loan_status',inplace = True)
df1.head(17)
# loandata.head()
```

Out[48]:

	emp_length	loan_status_clean	count_of_loan_issued
0	n/a	0	5319
1	n/a	1	11051

Almost half of 'n/a' records resulted into "bad loans"

In [49]:

```
# mean_emp_length_clean =
loandata[loandata.emp_length.notnull()].emp_length.mean()
# print(mean_emp_length_clean)
```

Cleaning Emp_length

Steps involve:- filling n/a with NaN. eliminating NaN by replacing it with 0. getting rid of < using regex. combining < 1 and 1 years together.

In [50]:

```
print(loandata.emp_length.value_counts())
```

```
10+ years    104925
2 years       28829
< 1 year      25992
3 years       25486
1 year        20807
5 years       18608
4 years       18600
n/a           16370
8 years       16365
7 years       16163
6 years       14710
9 years       12833
Name: emp_length, dtype: int64
```

In [51]:

```
loandata['emp_length'] = loandata.emp_length.str.replace('n/a','Not specifi
ed')
```

In [52]:

```
# eliminating n/a from emp_length

# loandata.replace('n/a', np.nan,inplace=True) #replacing n/a with np.nan
# loandata.emp_length_clean.fillna(value = 0,inplace=True)# filling 0 for n
p.nan
```

```
# loandata['emp_length_clean'].replace(to_replace='[^0-9]+', value='', inpl
ace=True, regex=True) # replacing < with empty space
# loandata['emp_length_clean'] = loandata.emp_length_clean.map(float)
# print(loandata.emp_length_clean.value_counts())
# loandata.emp_length_clean.unique().shape
# loandata.head(1)

loandata['emp_length_clean'] = loandata.emp_length.str.replace('+','')
loandata['emp_length_clean'] = loandata.emp_length_clean.str.replace('<',''
)
loandata['emp_length_clean'] = loandata.emp_length_clean.str.replace('years
','')
loandata['emp_length_clean'] = loandata.emp_length_clean.str.replace('year'
,'')
loandata['emp_length_clean'] = loandata.emp_length_clean.str.replace('Not s
pecified','99')
loandata['emp_length_clean'] = loandata.emp_length_clean.str.replace(' 1','
1')
print(loandata.emp_length_clean.value_counts())
```

```
10     104925
1       46799
2       28829
3       25486
5       18608
4       18600
99      16370
8       16365
7       16163
6       14710
9       12833
Name: emp_length_clean, dtype: int64
```

In [53]:

```
t1 = pd.crosstab(index=loandata["emp_length_clean"],columns=loandata["loan_
status_clean"], margins = True)
t1.columns = ["Bad Loan%","Good Loan%","Rtotal%"]
t1.index = ['Not spec', '<1  year', '2 years', '3 years', '4 years', '5 yea
rs', '6 years', ' 7 years' ,'8 years', '9 years', '10+ years','Ctotal%']
t0 = t1/t1.ix["Ctotal%","Rtotal%"]*100
t0
```

Out[53]:

	Bad Loan%	Good Loan%	Rtotal%
Not spec	3.651373	10.987588	14.638960
<1 year	7.183879	25.637184	32.821063
2 years	2.112059	6.905796	9.017855
3 years	1.879020	6.093128	7.972148
4 years	1.366645	4.451528	5.818173
5 years	1.380721	4.439954	5.820675
6 years	1.059783	3.541578	4.601361
7 years	1.155814	3.900053	5.055867
8 years	1.222755	3.896299	5.119054

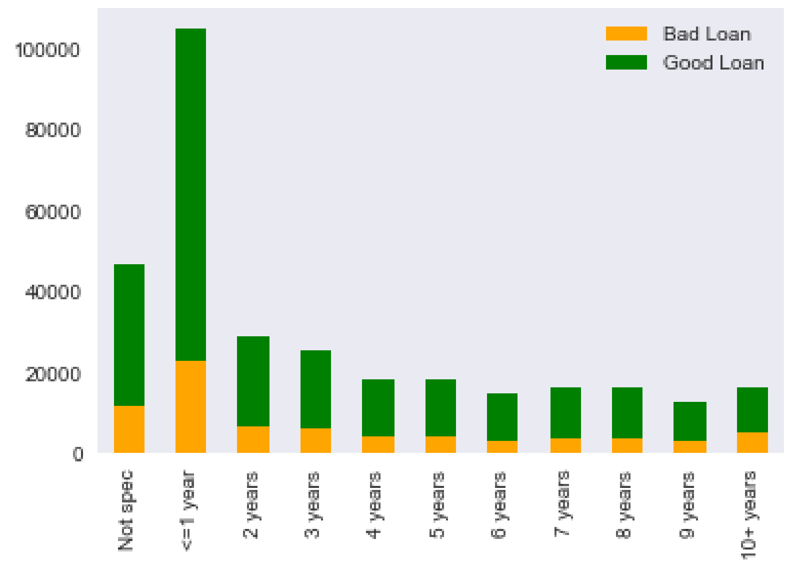
	Bad Loan%	Good Loan%	Rtotal%
9 years	0.973136	3.041090	4.014226
10+ years	1.663810	3.456808	5.120618
Ctotal%	23.648995	76.351005	100.000000

In [54]:

```
t2 = pd.crosstab(loandata['emp_length_clean'], loandata['loan_status_clean'
])
t2.columns = ["Bad Loan","Good Loan"]
t2.index = ['Not spec', '<=1 year', '2 years', '3 years', '4 years', '5 yea
rs', '6 years'
            , ' 7 years' ,'8 years', '9 years', '10+ years']

t2.plot(kind='bar', stacked=True, color=['orange','green'], grid=False)
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b13bdd8>
```



It's interesting to see the % of bad loans against 1 year pill vs others . We will explore whether or not it holds predictive power while building our model.

Analysing 'home_ownership'

"home ownership" status provided by the borrower during registration or obtained from the credit report. Values are: RENT, OWN, MORTGAGE, OTHER

In [55]:

```
loandata[ (loandata['home_ownership'].isnull())]##checking if the column ha
s null values
```

Out[55]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annua

In [56]:

In [56]:

```
loandata.home_ownership.unique().tolist()
```

Out[56]:

```
['MORTGAGE', 'RENT', 'OWN', 'ANY']
```

In [57]:

```
loandata['home_ownership']=loandata['home_ownership'].apply(lambda x:
'OTHER' if (x == 'NONE' or x=='ANY') else x)
```

In [58]:

```
home_ownership_dist=pd.DataFrame(loandata.home_ownership.value_counts()) #
1 omitted as it contains missing data
home_ownership_dist.reset_index(inplace=True)
home_ownership_dist.columns=['Home Ownership','Number of applicants']
print(home_ownership_dist)

# print(loandata_s.home_ownership.value_counts())
# loandata_s.home_ownership.unique().shape
```

```
  Home Ownership  Number of applicants
0       MORTGAGE                159786
1           RENT                127259
2            OWN                 32642
3          OTHER                     1
```

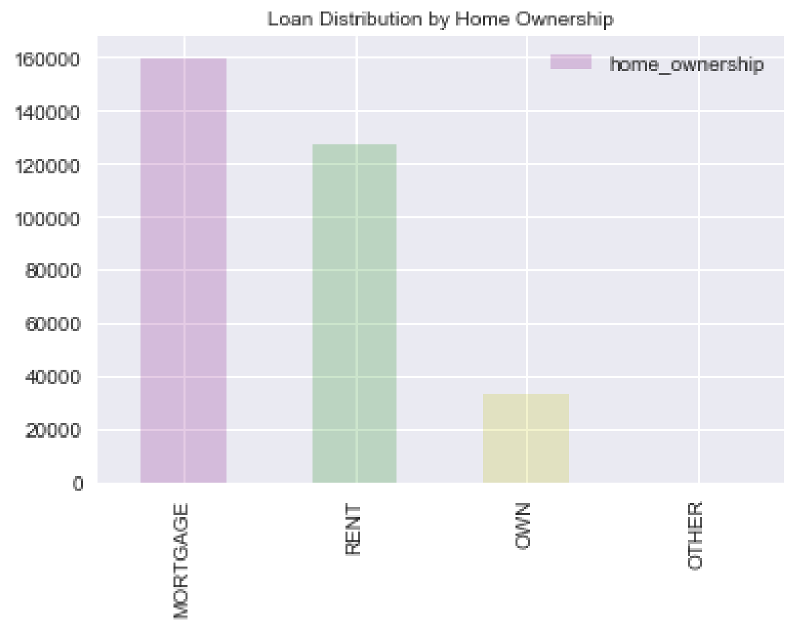
We have most number of application with ho_status 'Mortgage' status in this dataset and just one record with 'OTHER' which is odd. We will look at the loan_status for this record below.

In [59]:

```
colors = ['purple', 'green', 'y', 'pink','orange','blue']
loandata.home_ownership.value_counts().plot(kind='bar',alpha=.20,legend=Tru
e, color = colors)
plt.title('Loan Distribution by Home Ownership', fontsize = 10)
```

Out[59]:

```
&lt;matplotlib.text.Text at 0x1aa0e9b0&gt;
```



Lets look at the loan status for the "OTHER" category

In [60]:

```
pysql = lambda q: pdsql.sqldf(q, globals())

str1= """SELECT home_ownership, loan_status_clean, Avg(loan_amnt) as avg_lo
an ,count(loan_amnt) as NumOfApp
from loandata
where home_ownership = "OTHER"
and
(loan_status_clean = '0' or loan_status_clean = '1' )
group by home_ownership ,loan_status_clean
"""
df1 = pysql(str1)
df1.head(25)
```

Out[60]:

	home_ownership	loan_status_clean	avg_loan	NumOfApp
0	OTHER	1	5000.0	1

Cleaning home_ownership

In [61]:

```
loandata = loandata[loandata.home_ownership != 'OTHER']# Eliminating OTHER"

loandata.head(2)
print(loandata.home_ownership.value_counts())
```

```
MORTGAGE    159786
RENT        127259
OWN          32642
Name: home_ownership, dtype: int64
```

In [62]:

```
# loandata['home_ownership_clean'] =
loandata['home_ownership'].map({'MORTGAGE': 2, 'RENT': 3, 'OWN':1, 'OTHER':
4})
# loandata = loandata[loandata.home_ownership_clean != 4]# Eliminating OTHE
R"

loandata['home_ownership_clean'] = loandata['home_ownership'].map({'MORTGAG
E': 1, 'OWN': 1, 'RENT':0})
loandata["home_ownership_clean"] = loandata["home_ownership_clean"].apply(l
ambda home_ownership_clean: 0 if home_ownership_clean == 0 else 1)
```

We will assign OWN + MORTGAGE = '1' , 'RENT' = '0'

In [63]:

```
print(loandata.home_ownership_clean.value_counts())
```

```
1    192428
0    127259
```

```
Name: home_ownership_clean, dtype: int64
```

In [64]:

```
%matplotlib inline

import matplotlib
import numpy as np
import matplotlib.pyplot as plt
```

In [65]:

```
t1 = pd.crosstab(index=loandata["home_ownership"],columns=loandata["loan_st
atus_clean"], margins = True)
t1.columns = ["Bad Loan","Good Loan","Rtotal"]
t1.index = ["Mortgage", "Own","Rent", "Ctotal"]
t1
# t0 = t1/t1.ix["Ctotal","Rtotal"]*100
# t0
```

Out[65]:

	Bad Loan	Good Loan	Rtotal
Mortgage	32146	127640	159786
Own	8097	24545	32642
Rent	35360	91899	127259
Ctotal	75603	244084	319687

In [66]:

```
pysql = lambda q: pdsql.sqldf(q, globals())

str1= """SELECT home_ownership, loan_status_clean, count(loan_status_clean)
as NumOfApp
from loandata
group by home_ownership ,loan_status_clean
"""
df1 = pysql(str1)
df1.head(25)
```

Out[66]:

	home_ownership	loan_status_clean	NumOfApp
0	MORTGAGE	0	32146
1	MORTGAGE	1	127640
2	OWN	0	8097
3	OWN	1	24545
4	RENT	0	35360
5	RENT	1	91899

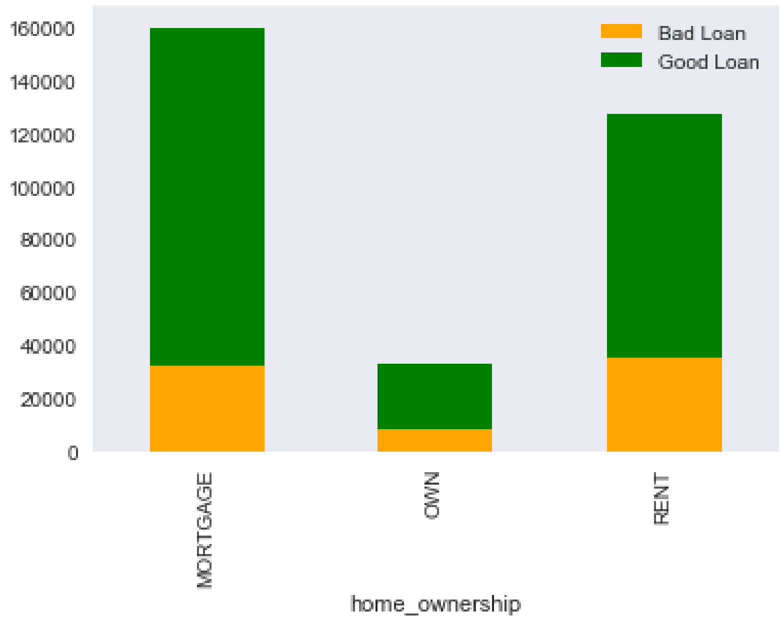
In [67]:

```
t2 = pd.crosstab(loandata['home_ownership'], loandata['loan_status_clean'])
```

```
t2 = pd.crosstab(loandata['home_ownership'], loandata['loan_status_clean'])
t2.columns = ["Bad Loan","Good Loan"]
# t2.index = ['MORTGAGE','Mortgage/Own']
t2.plot(kind='bar', stacked=True, color=['orange','green'], grid=False)
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1aa08ba8>
```



20%(32146/159786) of the loan with home_ownership = "MORTGAGE" has defaulted. 27% (35360/127259)of the loan with home_ownership = "RENT" has defaulted 24%(8097/32642)of the loans with home_ownership = "OWN" has defaulted. "Rent" status seems to have the biggest contribution to the bad loans as compared to "Mortgage,Own" categories.

In [68]:

```
# loandata['home_ownership_clean'] =
loandata['home_ownership'].map({'MORTGAGE': 2, 'OWN': 1, 'RENT':3})
# loandata["home_ownership_clean"] =
loandata["home_ownership_clean"].apply(lambda home_ownership_clean: 0 if ho
me_ownership_clean == 0 else 1)
```

Analysing "grade"

This is LC assigned loan grade: There are 7 loan grades ranging from A:F, A being the finest and F being the lowest grade. Lets look at distribution of loan_amnt against grades.

In [69]:

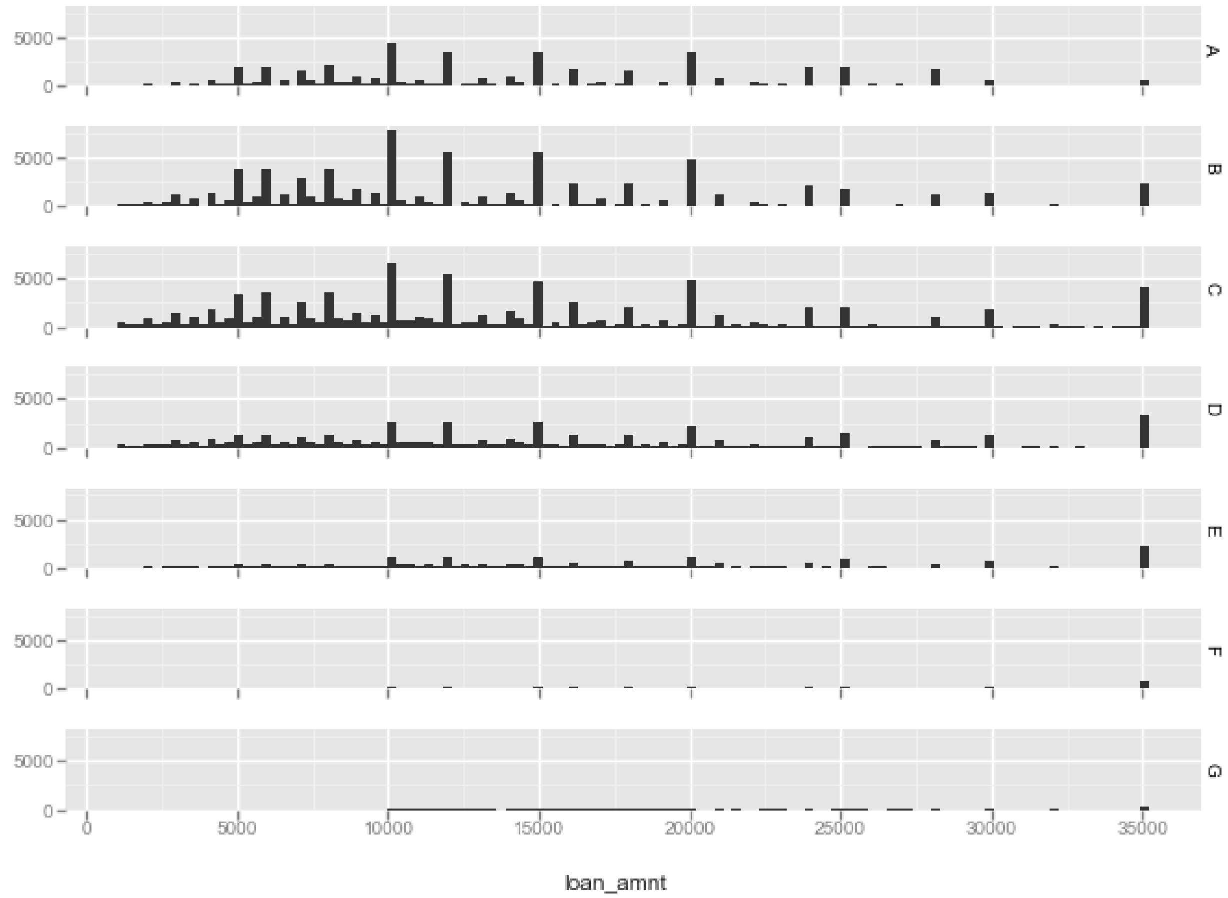
```
loandata[ (loandata['grade'].isnull())]##checking if the column has null va
lues
```

Out[69]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annua

In [70]:

```
from ggplot import *
# ggplot(loandata, aes(loan_amnt, col = grade)) + geom_histogram(bins =
50) + facet_grid(grade)
gg = ggplot(loandata, aes('loan_amnt',col='grade')) +
geom_histogram(binwidth=300)+facet_grid('grade')
print (gg)

#more loans have been allotted to the loan grade A,B,C,D compared to the lo
wer grades.
```




```
<ggplot: (-9223372036797379233)>
```

It appears more loans have been allotted to the loan grade A,B,C,D compared to the lower grades.

Cleaning grade

In [71]:

```
##Loan grade
loandata['grade_clean'] = loandata['grade'].map({'A':7,'B':6,'C':5,'D':4,'E'
:3,'F':2,'G':1})
```

In [72]:

```
loandata.head()
```

Out[72]:

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu
0	10400.0	10400.0	6.99%	8 years	Charged Off	MORTGAGE	A	5800

	loan_amnt	funded_amnt	int_rate	emp_length	loan_status	home_ownership	grade	annu
1	15000.0	15000.0	12.39%	10+ years	Fully Paid	RENT	C	7800
2	9600.0	9600.0	13.66%	10+ years	Fully Paid	RENT	C	6900
3	7650.0	7650.0	13.66%	< 1 year	Charged Off	RENT	C	5000
5	21425.0	21425.0	15.59%	6 years	Fully Paid	RENT	D	6380

In [73]:

```
t1 =
pd.crosstab(index=loandata["grade_clean"],columns=loandata["loan_status_clean"], margins = True)
t1.columns = ["Bad Loan","Good Loan","Rtotal"]
t1.index = ['1','2','3','4','5','6','7',"Ctotal"]

t1
```

Out[73]:

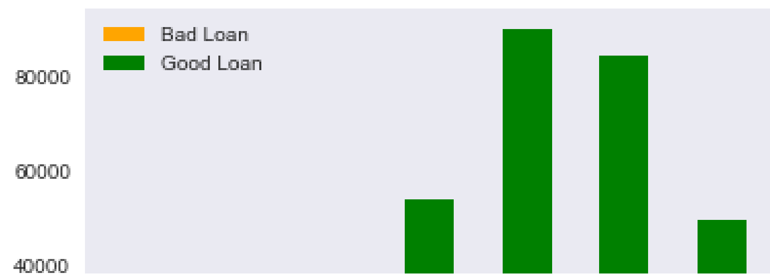
	Bad Loan	Good Loan	Rtotal
1	1345	1171	2516
2	4684	4874	9558
3	12386	16618	29004
4	18618	35484	54102
5	22493	67627	90120
6	12519	71965	84484
7	3558	46345	49903
Ctotal	75603	244084	319687

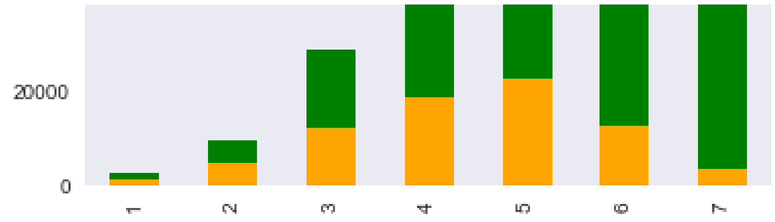
In [74]:

```
t2 = pd.crosstab(loandata['grade_clean'], loandata['loan_status_clean'])
t2.columns = ["Bad Loan","Good Loan"]
t2.index = ['1','2','3','4','5','6','7']
t2.plot(kind='bar', stacked=True, color=['orange','green'], grid=False)
```

Out[74]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x34ed5cf8>
```





Loan grade 1,2 & 3 appear to have 50:50 Good and bad loans.

In [75]:

```
# emp_length = loandata.emp_length_clean
# mean_emp_length_clean =
loandata[loandata.emp_length_clean.notnull()].emp_length_clean.mean()
# loandata.emp_length_clean.fillna(mean_emp_length_clean, inplace=True)

# grade = loandata.grade
# mean_grade_clean = loandata[loandata.grade.notnull()].grade_clean.mean()
# loandata.grade_clean.fillna(mean_grade_clean, inplace=True)
```

Model

I am going to use Logistic Regreassion which is an often used model in problems with binary target variables. Target variable in our case is Loan_status which is indeed a binary variable. It might not be the best approach, yet it definitely offers some insights in the data.

In [76]:

```
loandata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 319687 entries, 0 to 421093
Data columns (total 19 columns):
loan_amnt                319687 non-null float64
funded_amnt              319687 non-null float64
int_rate                 319687 non-null object
emp_length               319687 non-null object
loan_status              319687 non-null object
home_ownership           319687 non-null object
grade                    319687 non-null object
annual_inc               319687 non-null float64
verification_status      319687 non-null object
term                     319687 non-null object
dti                      319687 non-null float64
revol_bal                319687 non-null float64
total_acc                319687 non-null float64
loan_status_clean        319687 non-null int64
verification_status_clean    319687 non-null int64
term_clean               319687 non-null int64
emp_length_clean         319687 non-null object
home_ownership_clean     319687 non-null int64
grade_clean              319687 non-null int64
dtypes: float64(6), int64(5), object(8)
memory usage: 48.8+ MB
```

In [77]:

```
import statsmodels.api as sm
from sklearn import linear_model, datasets
from sklearn.linear_model import LinearRegression
import statsmodels.formula.api as smf
from scipy import stats
from sklearn.cross_validation import train_test_split
```

```
C:\Users\anands\AppData\Local\Continuum\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: DeprecationWarning:

This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are different from that of this module. This module will be removed in 0.20.
```

We will keep only selected columns and apply modelling techniques and explore the their predictive power against loan status. We call this df loan_2

In [78]:

```
loan_2 = loandata[['emp_length','emp_length_clean','loan_status_clean','home_ownership_clean','home_ownership','grade_clean','verification_status_clea
','term_clean']]
```

Logistic Regression

In [79]:

```
loan_2.shape
```

Out[79]:

```
(319687, 8)
```

As we touched upon data exploration and data munging above, we will try applying Logistic Regression algorithms to some of the cleaned columns and see how they impact the loan status. We will analyse Home_Ownership, Years of employed , Term, Grade, Emp_length and Verification_status.

home_ownership

We will apply logistic regression to the 'home_ownership' and predict the loan gets paid off or defaulted given the lender's home ownership status.' We will be creating dummy variables for individual statuses.

In [80]:

```
home_ownership = pd.get_dummies(loan_2.home_ownership)
loan_ho = loan_2.join(home_ownership)
loan_ho.head(1)
```

Out[80]:

	emp_length	emp_length_clean	loan_status_clean	home_ownership_clean	home_ownersh

	emp_length	emp_length_clean	loan_status_clean	home_ownership_clean	home_ownersh
0	8 years	8	0	1	MORTGAGE

In [81]:

```
X_Var_0 = ['MORTGAGE','OWN','RENT']
X_0 = loan_ho[X_Var_0]
```

In [82]:

```
X_0 = X_0.values
```

In [83]:

```
y_0 = loan_ho['loan_status_clean'].values
```

In [84]:

```
clf = linear_model.LogisticRegression()
```

In [85]:

```
model_0 = clf.fit(X_0,y_0)
print('intercept:', clf.intercept_)
print('coefficient:', clf.coef_[0])
```

```
intercept: [ 0.86489759]
coefficient: [ 0.464354    0.25904065  0.14150294]
```

In [86]:

```
model_0.score(X_0,y_0)
```

Out[86]:

```
0.76385566492402246
```

In [87]:

```
s= pd.DataFrame(list(zip(X_Var_0,model_0.coef_.T)))
s.columns = ["Status","Coef"]
# s.index = ['Ownership Status = Mortgage','Ownership Status = Own','Owners
hip Status = Rent']
s
```

Out[87]:

	Status	Coef
0	MORTGAGE	[0.464353997439]
1	OWN	[0.259040650821]
2	RENT	[0.141502939778]

Inference:-

Home ownership status marked as "Rent" have only 0.14 chance of paying back the loan where as "Mortgage" has 0.46 chance of paying off and "Own" = 0.25 chance of paying back loan.

EMP_LENGTH

In [88]:

```
emp_len= pd.get_dummies(loan_2.emp_length)
loan_emplen = loan_2.join(emp_len)
loan_emplen.head(1)
```

Out[88]:

	emp_length	emp_length_clean	loan_status_clean	home_ownership_clean	home_ownersh
0	8 years	8	0	1	MORTGAGE

In [89]:

```
X_Var_1 = ['< 1 year','2 years','3 years','4 years','5 years','6 years','7
years','8 years','9 years','10+ years','Not specified']
X_1 = loan_emplen[X_Var_1]
```

In [90]:

```
y_1 = loan_emplen['loan_status_clean'].values
```

In [91]:

```
model_1 = clf.fit(X_1,y_1)
```

In [92]:

```
model_1.score(X_1, y_1)
```

Out[92]:

```
0.7638556649240224 6
```

In [93]:

```
# pd.concat([pd.DataFrame(X_Var_1),pd.DataFrame(model_1.coef_.T)],axis = 1
)
s= pd.DataFrame(list(zip(X_Var_1,model_1.coef_.T)))
s.columns = ["Emp_length","Coef"]
# s.index = ['1','2','3','4','5','6','7','8','9','10']
s
```

Out[93]:

	Emp_length	Coef
0	< 1 year	[-0.0108478832174]
1	2 years	[0.0512277076293]
2	3 years	[0.0653339840592]
3	4 years	[0.0504631305282]
4	5 years	[0.0335182117012]
5	6 years	[0.0456009552521]

	Emp_length	Coef
6	7 years	[0.0648643222256]
7	8 years	[0.0386935922295]
8	9 years	[0.0158223785232]
9	10+ years	[0.12079719402]
10	Not specified	[-0.289418274895]

Inference:-

There does not seem to be striking variation in the coefficient values for the number of employment years between 2 to 9 years. They are hovering over the range of .01 to .05. However, applications with emp_length <= 1 year and also those which dint have the employment_length specified have -ve coefficients. THey are more likely to be defaulting. To my understanding, anybody not specifying the employment could be because they might not be bound with employment at the time loan was funded or could be in some other business. Their annual inc and verification status could be analysed further to draw any co-relation yet this feature does have predictive power to a certain extent as much intuitive it appears(a person would be continuing to pay off as lons as he/she is employed)

Verification Status and Term

In [94]:

```
X_Var_3 = ['verification_status_clean','term_clean']
X_3 = loan_2[X_Var_3]
```

In [95]:

```
X_3 = X_3.values
```

In [96]:

```
y_3 = loan_2['loan_status_clean'].values
```

In [97]:

```
model_3 = clf.fit(X_3,y_3)
print('intercept:', clf.intercept_)
print('coefficient:', clf.coef_[0])
```

```
intercept: [ 0.85136721]
coefficient: [-0.35284327  0.84230508]
```

In [98]:

```
model_3.score(X_3, y_3)
```

Out[98]:

```
0.76350930754143898
```

In [99]:

```
# pd.concat([pd.DataFrame(X_Var_3),pd.DataFrame(model_3.coef_.T)],axis = 1
)
s= pd.DataFrame(list(zip(X_Var_3,model_3.coef_.T)))
```

```
s.columns = ["Status","Coef"]
# s.index = ['Verified Status']
s
```

Out[99]:

	Status	Coef
0	verification_status_clean	[-0.352843271276]
1	term_clean	[0.842305079757]

Inference-

Verified Status indicates if income was verified by LC, there is a good chance of loan being defaulted if "Status" is not verified.

In [100]:

```
X_Var_4 = ['emp_length_clean','grade_clean']
X_4 = loan_2[X_Var_4]
```

In [101]:

```
X_4 = X_4.values
```

In [102]:

```
y_4 = loan_2['loan_status_clean'].values
```

In [103]:

```
model_4 = clf.fit(X_4,y_4)
```

In [104]:

```
model_4.score(X_4, y_4)
```

Out[104]:

```
0.76320275769737278
```

In [105]:

```
# pd.concat([pd.DataFrame(X_Var_5),pd.DataFrame(model_5.coef_.T)],axis = 1
)
pd.DataFrame(list(zip(X_Var_4,model_4.coef_.T)))
```

Out[105]:

	0	1
0	emp_length_clean	[-0.0046381801766]
1	grade_clean	[0.491244780449]

the coefficients are :

0.0255 for the lenght of employment

0.491 for the grade a loan received.

For every additional increase in the grade "G" to "F" or in our case "1" to "2" the chance of the loan being paid off increases by .469 which makes quite intuitive sense. LC assigns a high grade to a loan that they think is stable and low grade to a loan which is faulty. Conclusively as the grade for a loan increases the chance of the loan being paid off also increases by a coeff of .46

As for the number of employment years it's not the best predictor. For every each additional year that someone is employed the chance of that person paying back their loan increases only by 0.0248

Creating Train and Test Data-Set

Here we split the dataset into train:test in the ratio 7:3. And we will be using above variables to predict bad loans following below machine learning techniques. (1) Logistic regression (2) Random forest (3) k-Nearest neighbor (k=13) (4) Decision Tree

In [106]:

```
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
import pandas as pd
```

In [107]:

```
features = ['verification_status_clean'] +['home_ownership_clean']+['emp_le
ngth_clean']+['grade_clean']+['term_clean']
target = 'loan_status_clean'

# Now create an X variable (containing the features) and an y variable
(containing only the target variable)
X = loan_2[features]
y = loan_2[target]
```

In [108]:

```
# X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30)
```

Logistic Regression

In [109]:

```
# Define the model
lr = LogisticRegression()

# Define the splitter for splitting the data in a train set and a test set
splitter = StratifiedShuffleSplit(n_splits=1, test_size=0.33, random_state=
0)

# Loop through the splits (only one)
for train_indices, test_indices in splitter.split(X, y):
    # Select the train and test data
    X_train, y_train = X.iloc[train_indices], y.iloc[train_indices]
    X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]
```

```
    X_test, y_test = X.iloc[test_indices], y.iloc[test_indices]

    # Normalize the data
#     X_train = normalize(X_train)
#     X_test = normalize(X_test)

    # Fit and predict!
    lr_result= lr.fit(X_train, y_train)
    lr_pred = lr_result.predict(X_test)

#     # And finally: show the results
print(classification_report(y_test, lr_pred))
a = accuracy_score(y_test, lr_pred)
print("Accuracy:")
print(a*100)
```

```
             precision    recall  f1-score   support

          0       0.51      0.13      0.20     24949
          1       0.78      0.96      0.86     80548

avg / total       0.72      0.76      0.71    105497

Accuracy:
76.4315572955
```

The 0 classes (Defaulted/Charged Off loans) are predicted with .51 precision and .13 recall.

Random Forest

In [110]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
rf = RandomForestClassifier(n_estimators=10, min_samples_split=2)

rf_result=rf.fit(X_train,y_train)

rf_pred = rf_result.predict(X_test)
accuracy = accuracy_score(y_test, rf_pred)
accuracy
```

Out[110]:

```
0.76528242509265665
```

KNN

In [111]:

```
# ##K-nearest neighbor: let us try a range of k to see what might be the be
st k
# k_range=range(1,21)
# scores=[]
# for k in k_range:

#    knn = KNeighborsClassifier(n_neighbors=k)
```

```
#    knn_result=knn.fit(X_train,y_train)

#    knn_pred = knn.predict(X_test)

# #    scores.append(metrics.accuracy_score(y_test, knn_pred))
```

In [112]:

```
# plt.plot(k_range, scores)
# plt.xlabel('Value of k for KNN')
# plt.ylabel('Performance')
# plt.title('The effect of "k" in k-nearest neighbor')
```

In [113]:

```
knn = KNeighborsClassifier(n_neighbors=21)
knn_result=knn.fit(X_train,y_train)
knn_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, knn_pred)
accuracy
```

Out[113]:

```
0.75604993506924367
```

Decision Tree

In [114]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.cross_validation import train_test_split
```

In [115]:

```
X_V = ['emp_length_clean', 'grade_clean','verification_status_clean','home_ownership_clean','term_clean']
X = loan_2[X_V]

X = X.values
y = loan_2['loan_status_clean'].values
```

In [116]:

```
# X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33)
dt = GaussianNB()
dt_result = dt.fit(X_train,y_train)
dt_pred = dt_result.predict(X_test)
```

In [117]:

```
dt.score(X_train,y_train)
```

Out[117]:

```
0.74096829917363094
```

In [118]:

```
dt.score(X_test,y_test)
```

Out[118]:

0.73900679640179345

In [119]:

```python
from sklearn import metrics
def measure_performance(X,y,dt,show_accuracy=True,
show_classification_report=True, show_confusion_matrix=True):
    y_pred= dt.predict(X)
    if show_accuracy:
        print
("Accuracy:{0:.3f}".format(metrics.accuracy_score(y_test,dt_pred)),"\n")

    if show_classification_report:
        print ("Classification report")
        print (metrics.classification_report(y_test,dt_pred),"\n")

    if show_confusion_matrix:
        print ("Confusion matrix")
        print (metrics.confusion_matrix(y_test,dt_pred),"\n")

measure_performance(X_train,y_train,dt,show_accuracy=True,show_classificati
n_report=True,show_confusion_matrix=True)
```

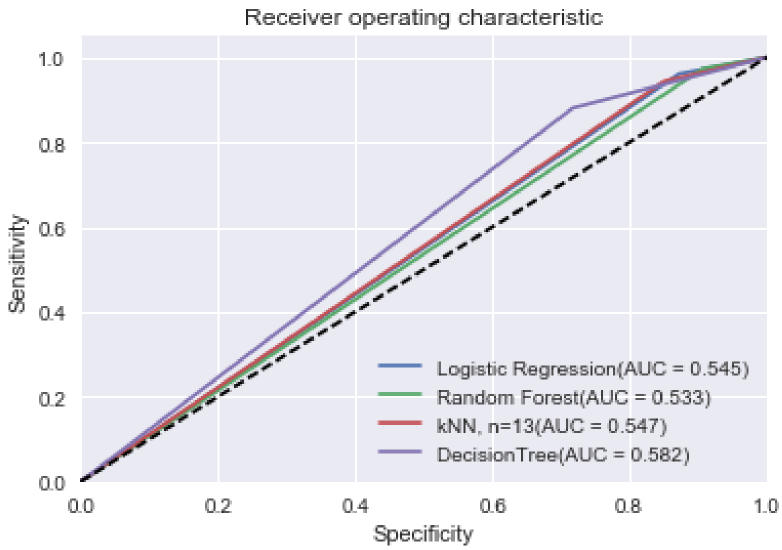
```
Accuracy:0.739

Classification report
             precision    recall  f1-score   support

          0       0.42      0.28      0.34     24949
          1       0.80      0.88      0.84     80548

avg / total       0.71      0.74      0.72    105497


Confusion matrix
[[ 7061 17888]
 [ 9646 70902]]
```

In [120]:

```python
from sklearn.metrics import roc_curve, auc
### ROC plot preparation
fpr, tpr, thresholds =roc_curve (y_test, lr_pred) #roc_curve(true
level,predicted outcome)
roc_auc = auc(fpr, tpr)

print("Area under the ROC curve -lr: %f" % roc_auc)

fpr2, tpr2, thresholds2 =roc_curve(y_test, rf_pred)
roc_auc2 = auc(fpr2,tpr2)
print("Area under the ROC curve -rf: %f" % roc_auc2)

fpr3, tpr3, thresholds3 =roc_curve(y_test, knn_pred)
roc_auc3 = auc(fpr3, tpr3)
print("Area under the ROC curve -knn: %f" % roc_auc3)

fpr4, tpr4, thresholds4 =roc_curve(y_test, dt_pred)
```

```
roc_auc4 = auc(fpr4, tpr4)
print("Area under the ROC curve -dt: %f" % roc_auc4)

### Plot ROC plots
plt.figure()
plt.plot(fpr,tpr,label='Logistic Regression(AUC = %0.3f)' % roc_auc)
plt.plot(fpr2,tpr2,label='Random Forest(AUC = %0.3f)' % roc_auc2)
plt.plot(fpr3,tpr3,label='kNN, n=13(AUC = %0.3f)' % roc_auc3)
plt.plot(fpr4,tpr4,label='DecisionTree(AUC = %0.3f)' % roc_auc4)
plt.xlim(0, 1)
plt.ylim(0, 1.05)
plt.plot([0, 1], [0, 1],'k--')
plt.xlabel('Specificity')
plt.ylabel('Sensitivity')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.show()
```

```
Area under the ROC curve -lr: 0.544587
Area under the ROC curve -rf: 0.533282
Area under the ROC curve -knn: 0.546769
Area under the ROC curve -dt: 0.581631
```



In [ ]: