VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

DATA MINING

ITE2006

Project Report

Topic

Peer-to-Peer Credit Lending System

Slot: F2+TF2

Faculty: Prabhavathy P

Group Members

Reshabh Kumar Jain 17BIT0048

Aradhya Mathur 17BIT0146

Abstract

Peer-to-peer credit lending system is the practice of lending money to individuals or businesses through online services that match lenders with borrowers. “Lending Club” is the world's largest peer-to-peer lending platform. The interest rates can be set by lenders who compete for the lowest rate on the reverse auction model or fixed by the intermediary company on the basis of an analysis of the borrower's credit. There is the risk of the borrower defaulting on the loans taken out from peer-lending websites.

Literature Review

1. **Loan Credibility Prediction System Based on Decision Tree Algorithm**

By: Sivasree M S, Rekha Sunny T

Decision Tree Induction Data Mining Algorithm is applied to predict the attributes relevant for credibility. A prototype of the model is described in this paper which can be used by the organizations in making the right decision to approve or reject the loan request of the customers

2. **Credit Risk Evaluating System Using Decision Tree – Neuro Based Model**

By: Ledisi G. Kabari

Decision making in a complex and dynamically changing environment of the present day demands a new technique of computational intelligence for building an equally adaptive, hybrid intelligent decision support system. In this paper, a Decision Tree- Neuro Based model was developed to handle loan granting decision support system. The system uses an integration of Decision Tree and Artificial Neural Networks with a hybrid of Decision Tree algorithm and Multilayer Feed-forward Neural Network with backpropagation learning algorithm to build up the proposed model.

3. **Determinants of Default in P2P Lending**

By: Carlos Serrano-Cinca ,Begoña Gutiérrez-Nieto ,Luz López-Palacios

The empirical study is based on loans' data collected from Lending Club (N = 24,449) from 2008 to 2014 that are first analysed by using univariate means tests and

survival analysis. Factors explaining default are loan purpose, annual income, current housing situation, credit history and indebtedness. Secondly, a logistic regression model is developed to predict defaults. The grade assigned by the P2P lending site is the most predictive factor of default, but the accuracy of the model is improved by adding other information, especially the borrower's debt level.

4. **Risk Assessment in Social Lending via Random Forests**

By: Milad Malekipirbazari, Vural Aksakall
They proposed a random forest (RF) based classification method for predicting borrower status. Their results on data from the popular social lending platform Lending Club (LC) indicate the RF-based method outperforms the FICO credit scores as well as LC grades in identification of good borrowers

5. **Neuro-Based Artificial Intelligence Model for Loan Decisions**

By: Shorouq Fathi Eletter, Saad Ghaleb Yaseen and Ghaleb Awad Elrefa
This study developed a proposed model that identifies artificial neural network as an enabling tool for evaluating credit applications to support loan decisions in the Jordanian Commercial banks. A multi-layer feed-forward neural network with backpropagation learning algorithm was used to build up the proposed model

6. **Ensemble Neural Network Strategy for Predicting Credit Default Evaluation**

By: A.R.Ghatge, P.P.Halkarnikar

In this paper, the architecture used the theory of artificial neural networks and business rules to correctly determine whether a customer is default or not. The Feed-forward back propagation neural network is used to predict the credit default. The results of applying the artificial neural networks methodology to classify credit risk based upon selected parameters show abilities of the network to learn the patterns.

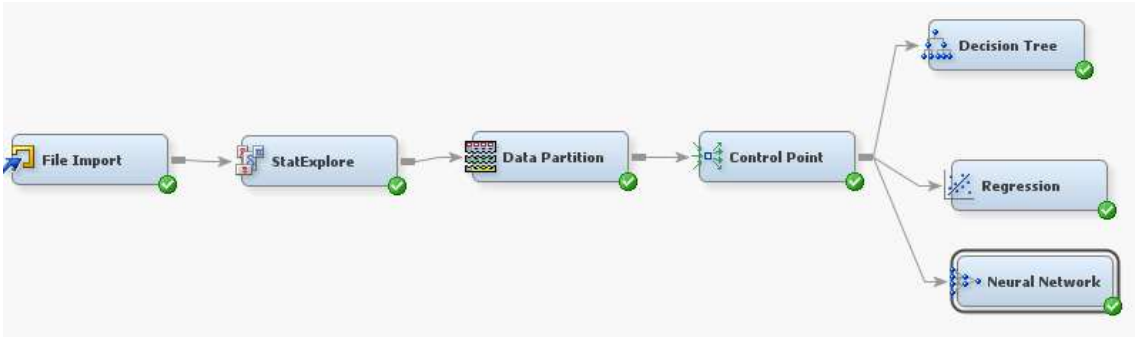
7. **Credit Risk Modeling using Multiple Regressions**

By: Maria Dimitriu, Ioana-Aurelia Oprea, Marian-Albert Scrieciu

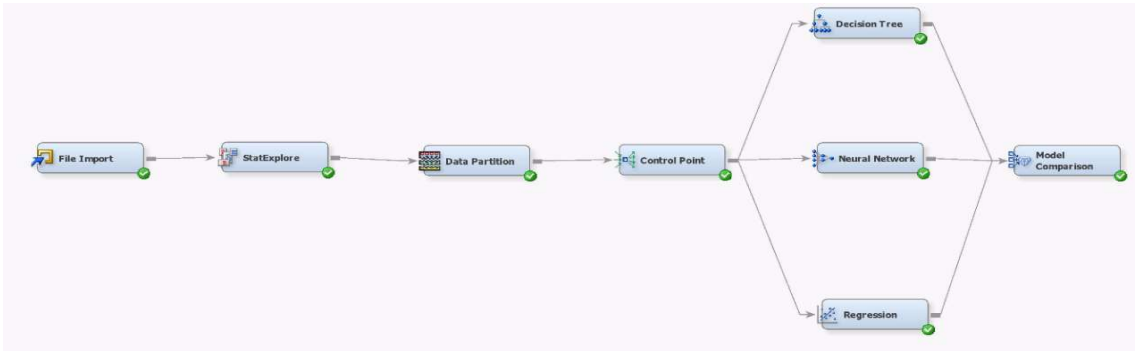
In classical theory, the risk is limited to mathematical expectation of losses that can occur when choosing one of the possible variants. For banks, risk is represented as losses arising from the completion of one or another decision. Bank risk is a phenomenon that occurs during the activity of banking operations and that causes negative effects for those activities: deterioration of business or record bank losses affecting functionality. It can be caused by internal or external causes, generated by the competitive environment. Here multilple regression is used

Framework

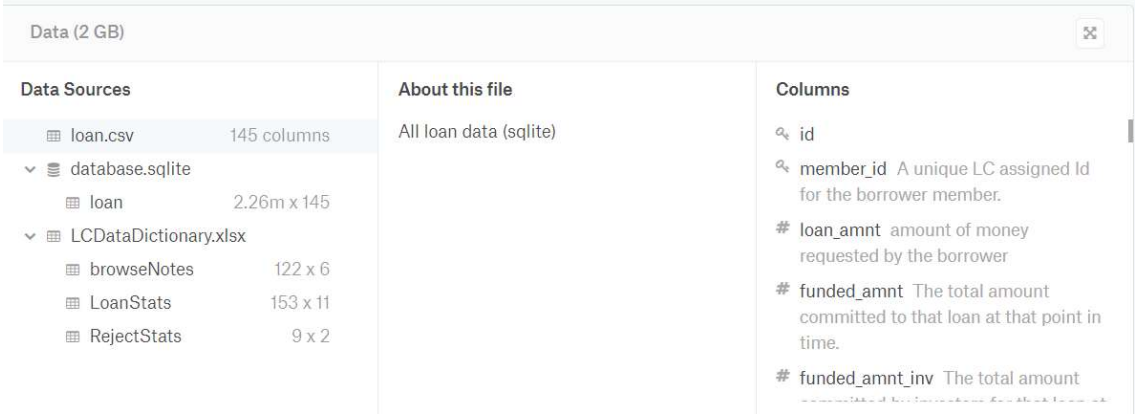
- Predicting whether a borrower would “charge-off”.

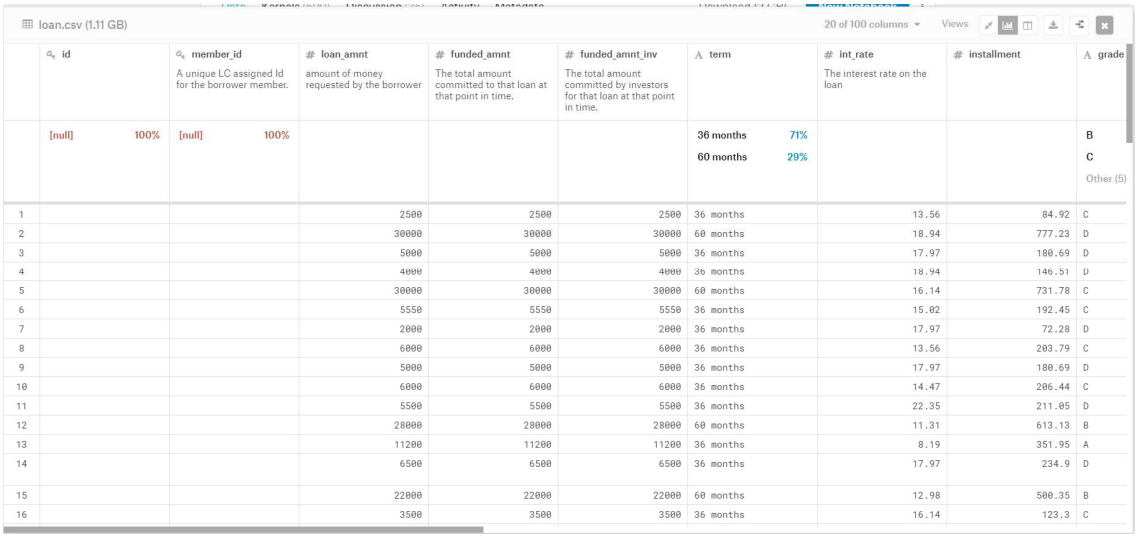


- Predicting whether a borrower will fall in the low, medium , high interest rate range.



Dataset





loan.csv (1.11 GB) — 20 of 100 columns — Views

	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade
		A unique LC assigned Id for the borrower member.	amount of money requested by the borrower	The total amount committed to that loan at that point in time.	The total amount committed by investors for that loan at that point in time.		The interest rate on the loan		
	[null] 100%	[null] 100%				36 months 71% 60 months 29%			B C Other (5)
1			2500	2500	2500	36 months	13.56	84.92	C
2			30000	30000	30000	60 months	18.94	777.23	D
3			5000	5000	5000	36 months	17.97	180.69	D
4			4000	4000	4000	36 months	18.94	146.51	D
5			30000	30000	30000	60 months	16.14	731.78	C
6			5550	5550	5550	36 months	15.02	192.45	C
7			2000	2000	2000	36 months	17.97	72.28	D
8			6000	6000	6000	36 months	13.56	203.79	C
9			5000	5000	5000	36 months	17.97	180.69	D
10			6000	6000	6000	36 months	14.47	206.44	C
11			5500	5500	5500	36 months	22.35	211.05	D
12			28000	28000	28000	60 months	11.31	613.13	B
13			11200	11200	11200	36 months	8.19	351.95	A
14			6500	6500	6500	36 months	17.97	234.9	D
15			22000	22000	22000	60 months	12.98	500.35	B
16			3500	3500	3500	36 months	16.14	123.3	C

Data Collection

Attribute	Explanation
loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
funded_amnt	The total amount committed to that loan at that point in time.
funded_amnt_inv	The total amount committed by investors for that loan at that point in time.

Attribute	Explanation
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
int_rate	Interest Rate on the loan
installment	The monthly payment owed by the borrower if the loan originates.
grade	LC assigned loan grade
sub_grade	LC assigned loan subgrade
emp_title	The job title supplied by the Borrower when applying for the loan.*
emp_length	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.
home_ownership	The home ownership status provided by the borrower during registration. Our values are: RENT,

Attribute	Explanation
	OWN, MORTGAGE, OTHER.
annual_inc	The self-reported annual income provided by the borrower during registration.
verification_status	
issue_d	The month which the loan was funded
loan_status	Current status of the loan
pymnt_plan	Indicates if a payment plan has been put in place for the loan
purpose	A category provided by the borrower for the loan request.
title	The loan title provided by the borrower
zip_code	The first 3 numbers of the zip code provided by the borrower in the loan application.

Attribute	Explanation
addr_state	The state provided by the borrower in the loan application
dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
delinq_2yrs	The number of 30+ days past-due incidences of delinquency in the borrower's credit file for the past 2 years
earliest_cr_line	The date the borrower's earliest reported credit line was opened
inq_last_6mths	The number of inquiries in past 6 months (excluding auto and mortgage inquiries)
open_acc	The number of open credit lines in the borrower's credit file.

Attribute	Explanation
pub_rec	Number of derogatory public records
revol_bal	Total credit revolving balance
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
total_acc	The total number of credit lines currently in the borrower's credit file
initial_list_status	The initial listing status of the loan. Possible values are – W, F
out_prncp	Remaining outstanding principal for total amount funded
out_prncp_inv	Remaining outstanding principal for portion of total amount funded by investors
total_pymnt	Payments received to date for total amount funded

Attribute	Explanation
total_pymnt_inv	Payments received to date for portion of total amount funded by investors
total_rec_prncp	Principal received to date
total_rec_int	Interest received to date
total_rec_late_fee	Late fees received to date
recoveries	post charge off gross recovery
collection_recovery_fee	post charge off collection fee
last_pymnt_d	Last month payment was received
last_pymnt_amnt	Last total payment amount received
last_credit_pull_d	The most recent month LC pulled credit for this loan
collections_12_mths_ex_med	Number of collections in 12 months excluding medical collections

Attribute	Explanation
policy_code	publicly available policy_code=1, new products not publicly available policy_code=2
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
acc_now_delinq	The number of accounts on which the borrower is now delinquent.
tot_coll_amt	Total collection amounts ever owed
tot_cur_bal	Total current balance of all accounts
total_rev_hi_lim	Total revolving high credit/credit limit
acc_open_past_24mths	Number of trades opened in past 24 months.
avg_cur_bal	Average current balance of all accounts

Attribute	Explanation
bc_open_to_buy	Total open to buy on revolving bankcards.
bc_util	Ratio of total current balance to high credit/credit limit for all bankcard accounts.
chargeoff_within_12_mths	Number of charge-offs within 12 months
delinq_amnt	The past-due amount owed for the accounts on which the borrower is now delinquent.
mo_sin_old_il_acct	
mo_sin_old_rev_tl_op	Months since oldest revolving account opened
mo_sin_rcnt_rev_tl_op	Months since most recent revolving account opened
mo_sin_rcnt_tl	Months since most recent account opened
mort_acc	Number of mortgage accounts.

Attribute	Explanation
mths_since_recent_bc	Months since most recent bankcard account opened.
mths_since_recent_inq	Months since most recent inquiry.
num_accts_ever_120_pd	Number of accounts ever 120 or more days past due
num_actv_bc_tl	Number of currently active bankcard accounts
num_actv_rev_tl	Number of currently active revolving trades
num_bc_sats	Number of satisfactory bankcard accounts
num_bc_tl	Number of bankcard accounts
num_il_tl	Number of installment accounts
num_op_rev_tl	Number of open revolving accounts
num_rev_accts	Number of revolving accounts

Attribute	Explanation
num_rev_tl_bal_gt_0	Number of revolving trades with balance >0
num_sats	Number of satisfactory accounts
num_tl_120dpd_2m	Number of accounts currently 120 days past due (updated in past 2 months)
num_tl_30dpd	Number of accounts currently 30 days past due (updated in past 2 months)
num_tl_90g_dpd_24m	Number of accounts 90 or more days past due in last 24 months
num_tl_op_past_12m	Number of accounts opened in past 12 months
pct_tl_nvr_dlq	Percent of trades never delinquent
percent_bc_gt_75	Percentage of all bankcard accounts > 75% of limit.
pub_rec_bankruptcies	Number of public record bankruptcies

Attribute	Explanation
tax_liens	Number of tax liens
tot_hi_cred_lim	Total high credit/credit limit
total_bal_ex_mort	Total credit balance excluding mortgage
total_bc_limit	Total bankcard high credit/credit limit
total_il_high_credit_limit	Total installment high credit/credit limit

Proposed Technology

Libraries Used

- **NumPy:** NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.
- **Pandas:** Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real

world data analysis in Python.

- **Matplotlib**: Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
- **Seaborn**: Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics
- **Ploty**: Plotly provides online graphing, analytics, and statistics tools for individuals and collaboration, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST

Algorithms Used

- ### Logistic Regression

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression[1] (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model,the log- odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can

vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a *logit*, from *logistic unit*, hence the alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a *constant* rate, with each independent variable having its own parameter; for a binary dependent variable this generalizes the odds ratio.

- **Random Forest**

  Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

  The first algorithm for random decision forests was created by Tin Kam Housing the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

- **XG Boost**

  XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

Imbalance Data

- **Anomaly Detection or Outlier Analysis:** In data mining, anomaly detection is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data
- **Over-sampling**: In signal processing, oversampling is the process of sampling a signal at a sampling frequency significantly higher than the Nyquist rate. Theoretically, a bandwidth-limited signal can be perfectly reconstructed if sampled at the Nyquist rate or above it.
- **Under-sampling**: In signal processing, under-sampling or bandpass sampling is a technique where one samples a bandpass-filtered signal at a sample rate below its Nyquist rate, but is still able to reconstruct the signal
- **SMOTE and ADASYN**: This is a statistical technique for increasing the number of cases in your dataset in a balanced way.

Codes and Output

In [1]:
```
import warnings
warnings.simplefilter('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

Load Dataset

The raw data are stored in separate files, ranging from 2007 until 2018. Possibly some years have been separated into 4 quarters.

In [2]:
```
# read raw data
data_2011 = pd.read_csv('./data/2011.csv')
data_2013 = pd.read_csv('./data/2013.csv')
data_2014 = pd.read_csv('./data/2014.csv')
data_2015 = pd.read_csv('./data/2015.csv')
data_2016_1 = pd.read_csv('./data/2016(1).csv')
data_2016_2 = pd.read_csv('./data/2016(2).csv')
data_2016_3 = pd.read_csv('./data/2016(3).csv')
data_2016_4 = pd.read_csv('./data/2016(4).csv')
data_2017_1 = pd.read_csv('./data/2017(1).csv')
data_2017_2 = pd.read_csv('./data/2017(2).csv')
data_2017_3 = pd.read_csv('./data/2017(3).csv')
data_2017_4 = pd.read_csv('./data/2017(4).csv')
data_2018 = pd.read_csv('./data/2018(1).csv')

# merge data from 2016 and 2017
data_2016 = pd.concat(objs=[data_2016_1, data_2016_2, data_2016_3, data_2016_4], axis=0)
data_2017 = pd.concat(objs=[data_2017_1, data_2017_2, data_2017_3, data_2017_4], axis=0)
```

In [3]:
```
# basic information
print('Year 2011:\t', data_2011.shape)
print('Year 2013:\t', data_2013.shape)
print('Year 2014:\t', data_2014.shape)
print('Year 2015:\t', data_2015.shape)
print('Year 2016:\t', data_2016.shape)
print('Year 2017:\t', data_2017.shape)
print('Year 2018:\t', data_2018.shape)
```

```
Year 2011:      (42538, 143)
Year 2013:      (188181, 143)
Year 2014:      (235629, 143)
Year 2015:      (421095, 143)
Year 2016:      (434407, 143)
Year 2017:      (443579, 143)
Year 2018:      (107864, 143)
```

Data Concatenation

After observation in depth, we notice that early date, especially the data before 2011, have some difference with the most recent data, such as the data in 2017 and 2018. Since the data before 2014 is only a small part of all data ($\sim 12\%$), in this project, we will only use the data from 2014 until 2018.

In [11]:
```
# add year information
data_2014['year'] = 2014
data_2015['year'] = 2015
data_2016_1['year'] = 2016
data_2016_2['year'] = 2016
data_2016_3['year'] = 2016
data_2016_4['year'] = 2016
data_2017_1['year'] = 2017
data_2017_2['year'] = 2017
data_2017_3['year'] = 2017
data_2017_4['year'] = 2017
data_2018['year'] = 2018
```

In [12]:
```
# concate data from 2014 until 2018
objs = [data_2014, data_2015, data_2016_1, data_2016_2, data_2016_3, data_2016_4,
        data_2017_1, data_2017_2, data_2017_3, data_2017_4, data_2018]
data = pd.concat(objs=objs, axis=0)
data.head()
```

Out[12]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	...	hardship_last_pay
0	10400	10400	10400.0	36 months	6.99%	321.08	A	A3	Truck Driver Delivery Personel	8 years	...	NaN
1	15000	15000	15000.0	60 months	12.39%	336.64	C	C1	MANAGEMENT	10+ years	...	NaN
2	9600	9600	9600.0	36 months	13.66%	326.53	C	C3	Admin Specialist	10+ years	...	NaN
3	7650	7650	7650.0	36 months	13.66%	260.20	C	C3	Technical Specialist	< 1 year	...	NaN
4	21425	21425	21425.0	60 months	15.59%	516.36	D	D1	Programming Analysis Supervisor	6 years	...	NaN

5 rows × 144 columns

Data Cleaning

In this notebook, we only do the very basic data cleaning steps: to remove the features with large amount of missing values.

In [14]:
```
# missing values
pd.options.display.max_rows = 150
missing = data.isnull().sum()
missing_ratio = missing / len(data)
missing_ratio = missing_ratio.reset_index()

# check the feature that has missing ration >= 0.2
missing_ratio = missing_ratio.rename(columns={'index': 'feature', 0: 'missing ratio'})
missing_ratio = missing_ratio.sort_values(by='missing ratio', ascending=False)
missing_ratio[missing_ratio['missing ratio'] >= 0.2]
```

Out[14]:

	feature	missing ratio
16	url	1.000000
132	orig_projected_additional_accrued_interest	0.996483
133	hardship_payoff_balance_amount	0.995612
124	deferral_term	0.995612
125	hardship_amount	0.995612
126	hardship_start_date	0.995612
127	hardship_end_date	0.995612
128	payment_plan_start_date	0.995612
129	hardship_length	0.995612
130	hardship_dpd	0.995612
123	hardship_status	0.995612
131	hardship_loan_status	0.995612
121	hardship_type	0.995612
134	hardship_last_payment_amount	0.995612
122	hardship_reason	0.995612
17	desc	0.990657
119	sec_app_mths_since_last_major_derog	0.988438
138	settlement_status	0.988106
142	settlement_term	0.988106

In [15]:
```
# it's safe to remove the features with missing ratio >= 0.3
columns = ['url', 'orig_projected_additional_accrued_interest', 'hardship_start_date',
           'hardship_end_date', 'payment_plan_start_date', 'hardship_length',
           'hardship_dpd', 'hardship_loan_status', 'hardship_payoff_balance_amount',
           'hardship_last_payment_amount', 'hardship_amount', 'deferral_term',
           'hardship_status', 'hardship_reason', 'hardship_type', 'desc',
           'sec_app_mths_since_last_major_derog', 'settlement_term',
           'debt_settlement_flag_date', 'settlement_date', 'settlement_amount',
           'settlement_percentage', 'settlement_status', 'sec_app_revol_util',
           'revol_bal_joint', 'sec_app_open_act_il', 'sec_app_earliest_cr_line',
           'sec_app_inq_last_6mths', 'sec_app_open_acc', 'sec_app_mort_acc',
           'sec_app_num_rev_accts', 'sec_app_chargeoff_within_12_mths',
           'sec_app_collections_12_mths_ex_med', 'verification_status_joint',
           'dti_joint', 'annual_inc_joint', 'mths_since_last_record',
           'mths_since_recent_bc_dlq', 'mths_since_last_major_derog',
           'mths_since_recent_revol_delinq', 'mths_since_last_delinq', 'next_pymnt_d',
           'il_util', 'mths_since_rcnt_il', 'all_util', 'open_acc_6m', 'total_cu_tl',
           'inq_last_12m', 'open_act_il', 'open_il_12m', 'max_bal_bc', 'open_rv_24m',
           'open_rv_12m', 'total_bal_il', 'open_il_24m', 'inq_fi']
```

In [16]:
```
# drop features with too many missing values
data = data.drop(labels=columns, axis=1)

# save to local disk
data.to_csv('./data/data_2014_2018.csv', index=False)

data.head()
```

Out[16]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_title	emp_length	...	pub_rec_bankrupt
0	10400	10400	10400.0	36 months	6.99%	321.08	A	A3	Truck Driver Delivery Personel	8 years	...	0
1	15000	15000	15000.0	60 months	12.39%	336.64	C	C1	MANAGEMENT	10+ years	...	0
2	9600	9600	9600.0	36 months	13.66%	326.53	C	C3	Admin Specialist	10+ years	...	0
3	7650	7650	7650.0	36 months	13.66%	260.20	C	C3	Technical Specialist	< 1 year	...	0
4	21425	21425	21425.0	60 months	15.59%	516.36	D	D1	Programming Analysis Supervisor	6 years	...	0

5 rows × 88 columns

Data Cleaning

1. Features collected after loan is issued

For this problem, we will assume that our model will run at the moment one begins to apply for the loan. Thus, there should be no information about user's payment behaviors.

In [4]:
```
ongo_columns = ['funded_amnt', 'funded_amnt_inv', 'issue_d', 'pymnt_plan', 'out_prncp',
                'out_prncp_inv', 'total_pymnt', 'total_pymnt_inv', 'total_rec_prncp',
                'policy_code', 'total_rec_int', 'total_rec_late_fee', 'recoveries',
                'collection_recovery_fee', 'last_pymnt_d', 'last_pymnt_amnt',
                'last_credit_pull_d', 'hardship_flag', 'disbursement_method',
                'debt_settlement_flag']
data = data.drop(labels=ongo_columns, axis=1)

data.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1642574 entries, 0 to 1642573
Columns: 68 entries, loan_amnt to year
dtypes: float64(14), int64(37), object(17)
memory usage: 852.2+ MB
```

2. Parsing `loan_status`

In [5]:
```
data['loan_status'].value_counts()
```

Out[5]:
```
Current               808178
Fully Paid            623342
Charged Off           173491
Late (31-120 days)     19438
In Grace Period        13526
Late (16-30 days)       4575
Default                   24
Name: loan_status, dtype: int64
```

3. Split into training and test dataset

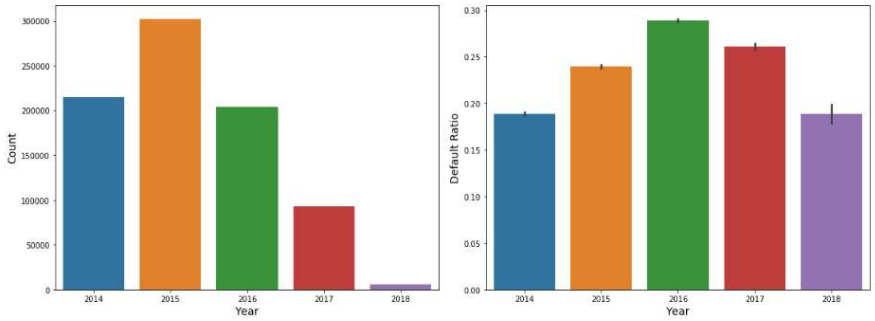
After above procedures, we have reduced the dataset size from $1,642,574$ to $820,870$, and the features from 88 to 68 (including the `target`).

In [8]:
```
# calculate the number of records for each year
year_count = data.groupby('year')['target'].count().reset_index()
year_count = year_count.rename(columns={'target': 'counts'})
year_count['ratio'] = year_count['counts'] / len(data)
year_count
```

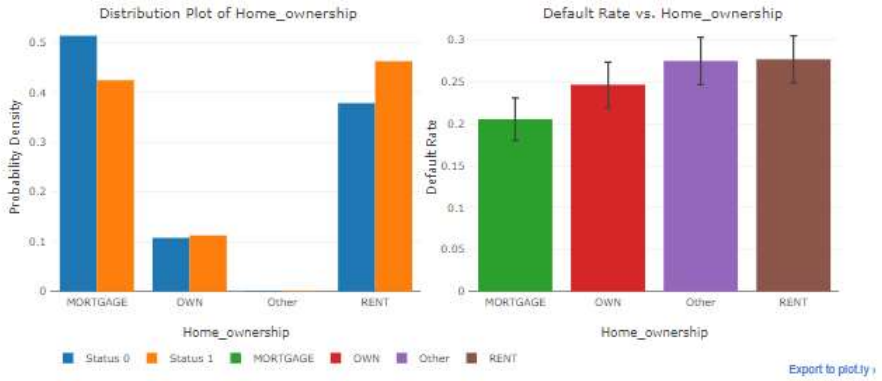
Out[8]:

	year	counts	ratio
0	2014	215210	0.262173
1	2015	302540	0.368560
2	2016	204393	0.248996
3	2017	92578	0.112780
4	2018	6149	0.007491

In [9]:
```
# visualize the time effect
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
sns.countplot(x='year', data=data, ax=ax[0])
ax[0].set_xlabel('Year', fontsize=14)
ax[0].set_ylabel('Count', fontsize=14)
sns.barplot(x='year', y='target', data=data, ax=ax[1])
ax[1].set_xlabel('Year', fontsize=14)
ax[1].set_ylabel('Default Ratio', fontsize=14)
plt.tight_layout()
plt.show()
```



In [7]:
```
# home_ownership
feature = 'home_ownership'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```



Invalid Data

In [5]:
```
# there are some problem with `int_rate` and `revol_util`
print(train['int_rate'].values[:3], '\t', test['int_rate'].values[:3])
print(train['revol_util'].values[:3], '\t', train['revol_util'].values[:3])
```

```
['6.99%' '12.39%' '13.66%']      ['6.99%' '12.74%' '6.99%']
['31.60%' '29%' '59.40%']        ['31.60%' '29%' '59.40%']
```

In [6]:
```
def str_parser(x):
    """ function to parse `int_rate` and `revol_util` """
    if '%' in str(x):
        return float(str(x).replace('%', ''))
    return x
```

In [7]:
```
train['int_rate'] = train['int_rate'].apply(str_parser)
train['revol_util'] = train['revol_util'].apply(str_parser)
test['int_rate'] = test['int_rate'].apply(str_parser)
test['revol_util'] = test['revol_util'].apply(str_parser)
```

Missing Values

1. Categorical Variable

In [8]:
```
# `emp_title` contains too many categories, one choice is to delete it
train = train.drop(labels='emp_title', axis=1)
test = test.drop(labels='emp_title', axis=1)
```

In [9]:
```
# for `emp_length`, since the order matters, we will fill the missing value with mode
emp_length_mode = train['emp_length'].mode()[0]
train['emp_length'] = train['emp_length'].fillna(value=emp_length_mode)
test['emp_length'] = test['emp_length'].fillna(value=emp_length_mode)
```

In [10]:
```
# for `title`, we will fill the missing value with `missing`
train['title'] = train['title'].fillna(value='missing')
test['title'] = test['title'].fillna(value='missing')
```

2. Numerical Variable

In [11]:
```
# for `dti`, there is one value with `dti`= -1, only in training data, just remove it
train = train[train['dti'] != -1]

# fill the missing value with the median to make it more robust
dti_median = train['dti'].median()
train['dti'] = train['dti'].fillna(value=dti_median)
test['dti'] = test['dti'].fillna(value=dti_median)
```

Feature Parsing

In [23]:
```
data_null[data_null['dtype'] == object]
```

Out[23]:

	feature	train miss	test miss	dtype	null mean	not null mean	train unique	test unique
1	term	0.000000	0.000000	object	NaN	0.238444	2	2
2	int_rate	0.000000	0.000000	object	NaN	0.238444	211	92
4	grade	0.000000	0.000000	object	NaN	0.238444	7	7
5	sub_grade	0.000000	0.000000	object	NaN	0.238444	35	35
6	emp_title	0.058919	0.073901	object	0.313223	0.233763	184018	35913
7	emp_length	0.057162	0.072807	object	0.316263	0.233727	12	12
8	home_ownership	0.000000	0.000000	object	NaN	0.238444	4	5
10	verification_status	0.000000	0.000000	object	NaN	0.238444	3	3
11	purpose	0.000000	0.000000	object	NaN	0.238444	14	13
12	title	0.015882	0.000000	object	0.372482	0.236281	1970	12
13	zip_code	0.000000	0.000000	object	NaN	0.238444	934	872
14	addr_state	0.000000	0.000000	object	NaN	0.238444	51	50
17	earliest_cr_line	0.000000	0.000000	object	NaN	0.238444	690	637
22	revol_util	0.000507	0.000831	object	0.226776	0.238450	1240	1077
24	initial_list_status	0.000000	0.000000	object	NaN	0.238444	2	2
26	application_type	0.000000	0.000000	object	NaN	0.238444	2	2

1. home_ownership

for home_ownership, test data have 5 categories while training data only have 4

In [24]:
```
test['home_ownership'].value_counts()
```

Out[24]:
```
MORTGAGE    49505
RENT        36736
OWN         12359
ANY           125
NONE            2
Name: home_ownership, dtype: int64
```

2. grade and sub_grade

Since for grade and sub_grade, the order matters, we will user label-encoder to encode it

In [27]:
```
grade_encoder = LabelEncoder()
grade_encoder = grade_encoder.fit(train['grade'])

train['grade'] = grade_encoder.transform(train['grade'])
test['grade'] = grade_encoder.transform(test['grade'])
```

In [28]:
```
sub_grade_encoder = LabelEncoder()
sub_grade_encoder = grade_encoder.fit(train['sub_grade'])

train['sub_grade'] = grade_encoder.transform(train['sub_grade'])
test['sub_grade'] = grade_encoder.transform(test['sub_grade'])
```

3. emp_length,

For emp_length, there are some inherent order

In [29]:
```
# let's manually endoce it
emp_length_map = {'< 1 year': 0,
                  '1 year': 1,
                  '2 years': 2,
                  '3 years': 3,
                  '4 years': 4,
                  '5 years': 5,
                  '6 years': 6,
                  '7 years': 7,
                  '8 years': 9,
                  '9 years': 9,
                  '10+ years': 10}
```

In [30]:
```
train['emp_length'] = train['emp_length'].apply(lambda x: emp_length_map[x])
test['emp_length'] = test['emp_length'].apply(lambda x: emp_length_map[x])
```

4. title

For title, we need to manually encode them to reduce the categorical level

In [31]:
```
# change into lower case
train['title'] = train['title'].apply(str.lower)
test['title'] = test['title'].apply(str.lower)
```

In [32]:
```
lists = ['debt consolidation', 'credit card refinancing', 'business', 'vacation',
         'home improvement', 'major purchase', 'medical expenses', 'car financing',
         'moving and relocation', 'home buying', 'green loan', 'consolidation']

train.loc[~train['title'].isin(lists), 'title'] = 'other'
test.loc[~test['title'].isin(lists), 'title'] = 'other'
```

Final Note

Let's check the cleaned dataset again

In [38]:
```
# get missing information
train_null = train.isnull().sum() / len(train)
train_null = train_null.reset_index()
train_null = train_null.rename(columns={'index': 'feature', 0: 'train miss'})

test_null = test.isnull().sum() / len(test)
test_null = test_null.reset_index()
test_null = test_null.rename(columns={'index': 'feature', 0: 'test miss'})

# merge train and test null information
data_null = pd.merge(left=train_null, right=test_null, how='outer')
data_null = data_null.fillna(value=0)

feature_list = data_null['feature']
null_rate = []
not_null_rate = []
category = []
train_unique = []
test_unique = []
for feature in feature_list:
    null_rate.append(train[train[feature].isnull()]['target'].mean())
    not_null_rate.append(train[~train[feature].isnull()]['target'].mean())
    category.append(train[feature].dtype)
    train_unique.append(len(train[feature].unique()))
    test_unique.append(len(test[feature].unique()))

data_null['dtype'] = category
data_null['null mean'] = null_rate
data_null['not null mean'] = not_null_rate
data_null['train unique'] = train_unique
data_null['test unique'] = test_unique

data_null = data_null.reset_index(drop=True)

data_null[['feature', 'train miss', 'test miss', 'dtype', 'null mean',
           'not null mean', 'train unique', 'test unique']]
```

Out[38]:

	feature	train miss	test miss	dtype	null mean	not null mean	train unique	test unique
0	loan_amnt	0.0	0.0	int64	NaN	0.238445	1482	1471
1	term	0.0	0.0	object	NaN	0.238445	2	2
2	int_rate	0.0	0.0	float64	NaN	0.238445	211	92
3	grade	0.0	0.0	int64	NaN	0.238445	7	7
4	sub_grade	0.0	0.0	int64	NaN	0.238445	35	35
5	emp_length	0.0	0.0	int64	NaN	0.238445	10	10
6	home_ownership	0.0	0.0	object	NaN	0.238445	4	4
7	annual_inc	0.0	0.0	float64	NaN	0.238445	41183	8508

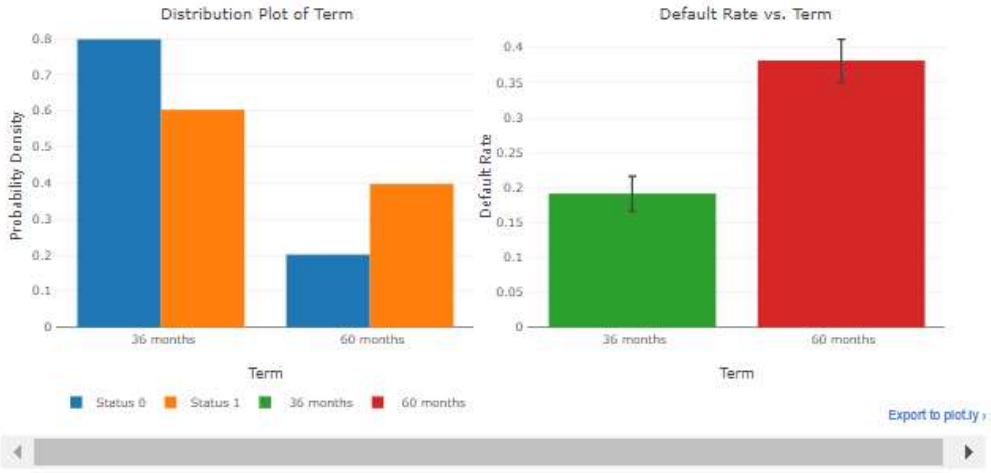
Visualization

In [5]:
```
# define categorical and numerical features
cat_features = ['term', 'home_ownership', 'verification_status', 'purpose',
                'title', 'addr_state', 'initial_list_status', 'application_type',
                'grade', 'sub_grade']

num_features = ['loan_amnt', 'loan_to_inc', 'int_rate', 'installment_ratio', 'emp_length',
                'annual_inc', 'dti', 'delinq_2yrs', 'inq_last_6mths', 'open_acc', 'pub_rec',
                'revol_bal', 'revol_util', 'total_acc', 'collections_12_mths_ex_med',
                'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim',
                '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_inq',
                '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', 'credit_length']

features = cat_features + num_features

# define numerical and categorical features
print('Categorical feature:\t', len(cat_features))
print('Numerical feature:\t', len(num_features))
print('Total feature:\t\t', len(features))
```

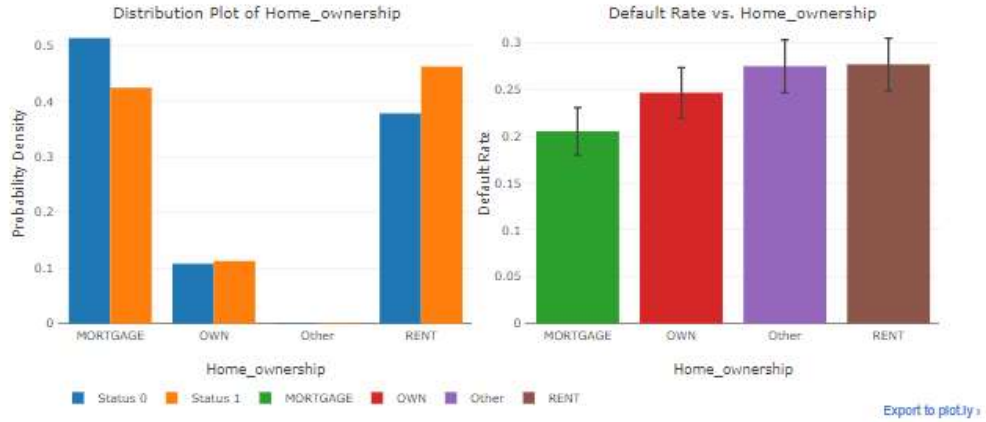
```
Categorical feature:    10
Numerical feature:      55
Total feature:          65
```

Categorical Features

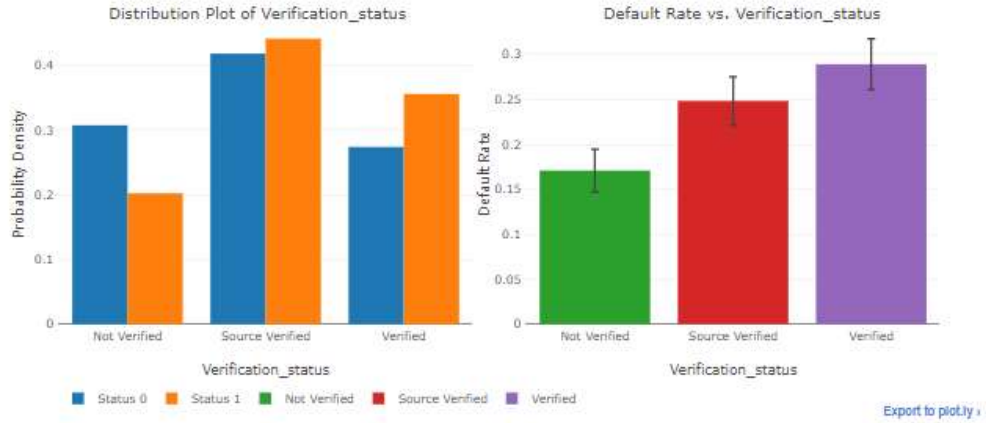
In [6]:
```
# term
feature = 'term'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```



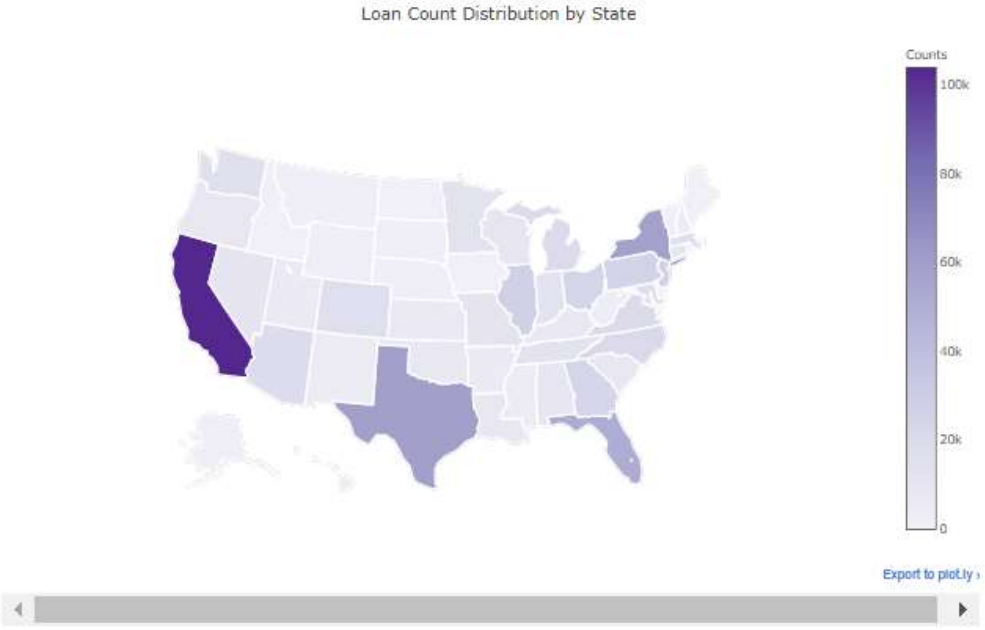
In [7]:
```
# home_ownership
feature = 'home_ownership'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```



In [8]:
```
# verification_status
feature = 'verification_status'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```

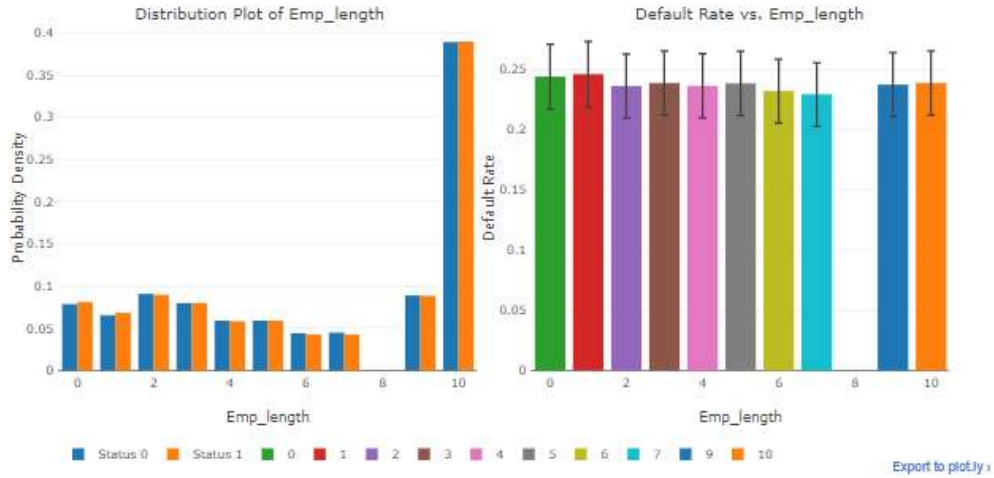


In [11]:
```
# addr_state
state_count = train.groupby('addr_state')['target'].count().reset_index()
state_count = state_count.sort_values(by='target', ascending=False)

# visualization
scl = [[0.0, 'rgb(242,240,247)'], [0.2, 'rgb(218,218,235)'],
       [0.4, 'rgb(188,189,220)'], [0.6, 'rgb(158,154,200)'],
       [0.8, 'rgb(117,107,177)'], [1.0, 'rgb(84,39,143)']]

data = [dict(type='choropleth', colorscale=scl, autocolorscale=False,
        locations=state_count['addr_state'], z=state_count['target'],
        locationmode='USA-states', colorbar=dict(title='Counts'),
        marker=dict(line=dict(color = 'rgb(255,255,255)', width=2)))]

geo = dict(scope='usa', projection=dict(type='albers usa'),
           showlakes=True, lakecolor='rgb(255, 255, 255)')

layout = dict(title='Loan Count Distribution by State', geo=geo,
              margin=go.Margin(l=50, r=50, b=50, t=40, pad=4),
              width=1000, height=600)

fig = dict(data=data, layout=layout)
iplot(fig)
```

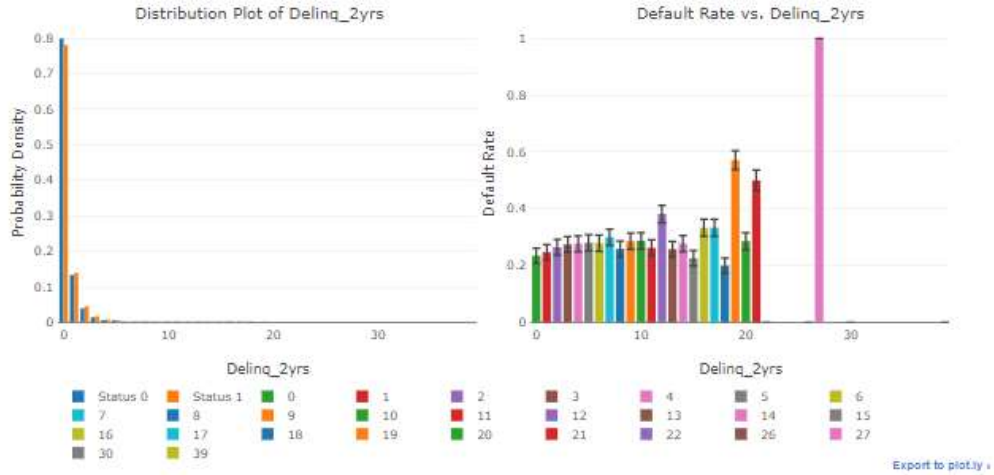


Discrete Features

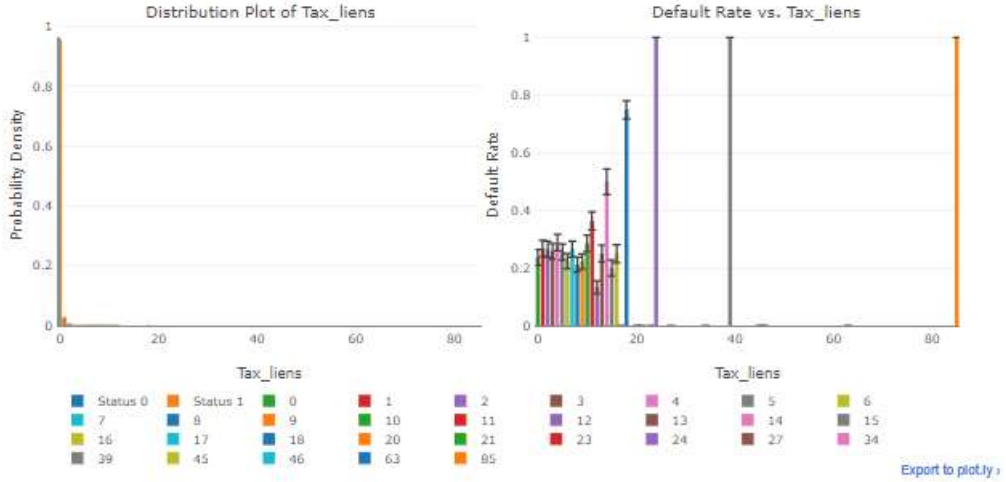
In [17]:
```
# emp_length
feature = 'emp_length'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=500))
```



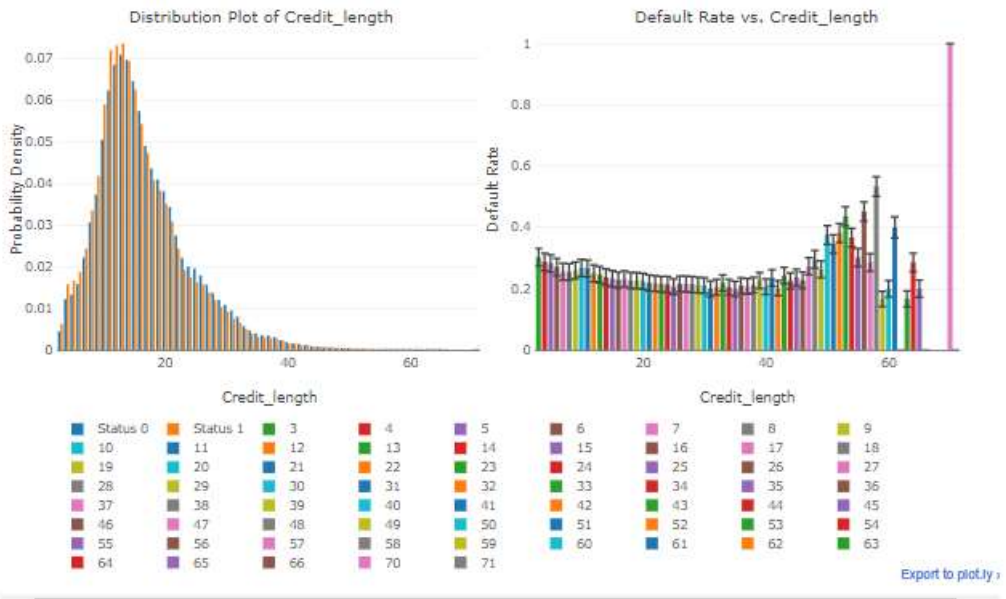
In [18]:
```
# delinq_2yrs
feature = 'delinq_2yrs'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=500))
```



In [42]:
```
# tax_liens
feature = 'tax_liens'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=500))
```



In [43]:
```
# credit_length
feature = 'credit_length'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=600))
```



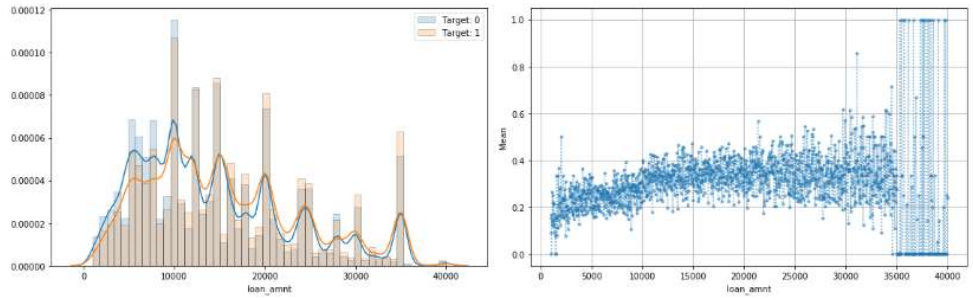
Visualization

In [5]:
```
# define categorical and numerical features
cat_features = ['term', 'home_ownership', 'verification_status', 'purpose',
                'title', 'addr_state', 'initial_list_status', 'application_type',
                'grade', 'sub_grade']

num_features = ['loan_amnt', 'loan_to_inc', 'int_rate', 'installment_ratio', 'emp_length',
                'annual_inc', 'dti', 'delinq_2yrs', 'inq_last_6mths', 'open_acc', 'pub_rec',
                'revol_bal', 'revol_util', 'total_acc', 'collections_12_mths_ex_med',
                'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim',
                '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_inq',
                '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', 'credit_length']

features = cat_features + num_features

# define numerical and categorical features
print('Categorical feature:\t', len(cat_features))
print('Numerical feature:\t', len(num_features))
print('Total feature:\t\t', len(features))
```

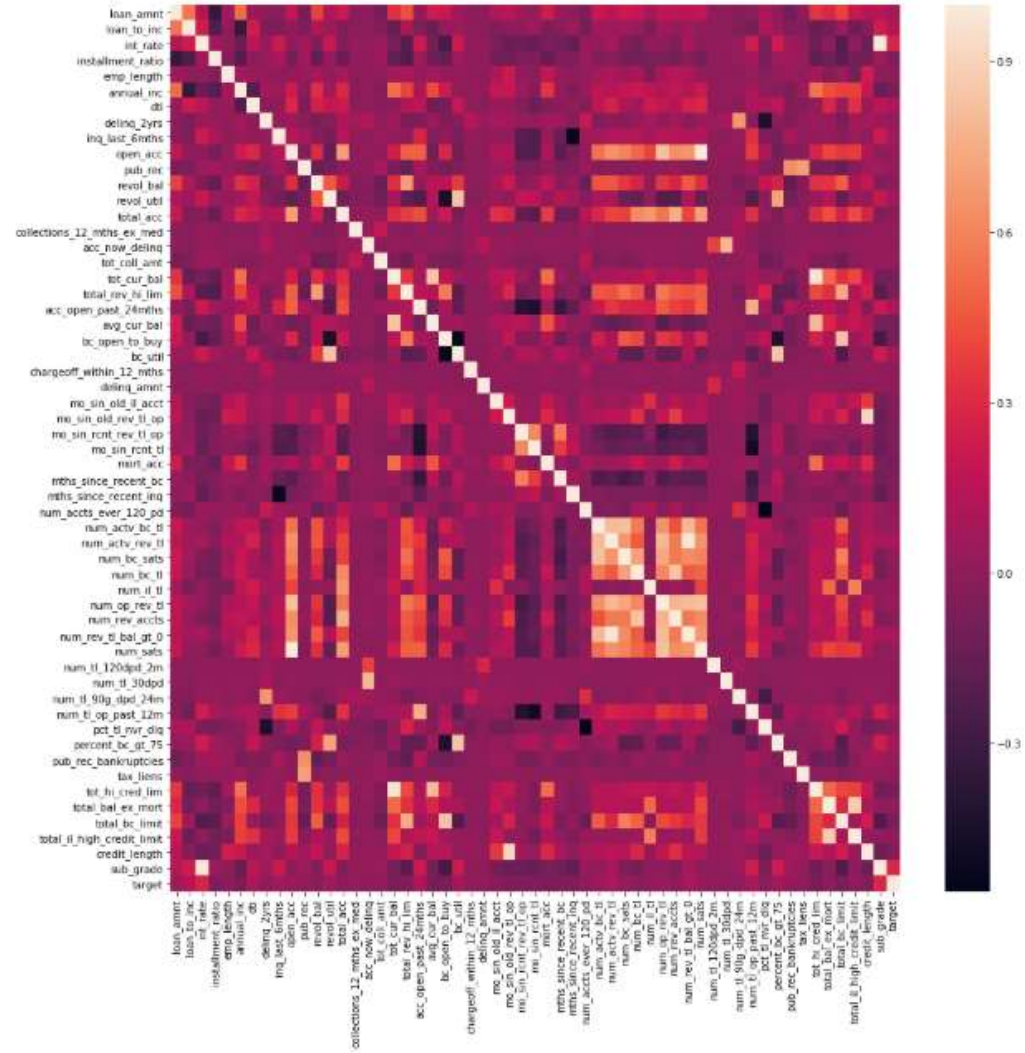
```
Categorical feature:    10
Numerical feature:      55
Total feature:          65
```

2. Numerical Variables

In [6]:
```
# Loan_amnt
feature = 'loan_amnt'
fig, ax = plot_numerical(train, feature=feature, figsize=(16, 5))
plt.show()
```



In [34]:
```
# calculate the pair-wise correlation
subset = num_features + ['sub_grade', 'target']
correlation = train[subset].corr()

# visualize the pair-wise correlation
fig, ax = plt.subplots(figsize=(16, 16))
sns.heatmap(correlation, ax=ax)
plt.show()
```



Visualization

The original data set contains **65** different features, and a binary label for `target`. Among all the features, there are **10** categorical features and **55** numerical features. However, some numerical features are essentially discrete.

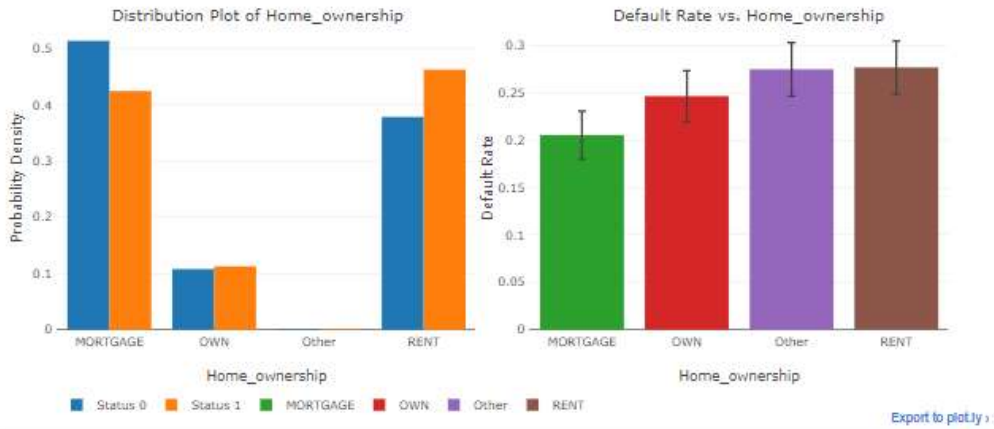
Below, categorical features and discrete features are visualized in same style. And continuous features are visualized in another style.

I. Categorical Features

In [4]:
```
# term
feature = 'term'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```

Export to plot.ly »

In [5]:
```
# home_ownership
feature = 'home_ownership'
iplot(discrete_plot(data=train, feature=feature, width=1000, height=450))
```

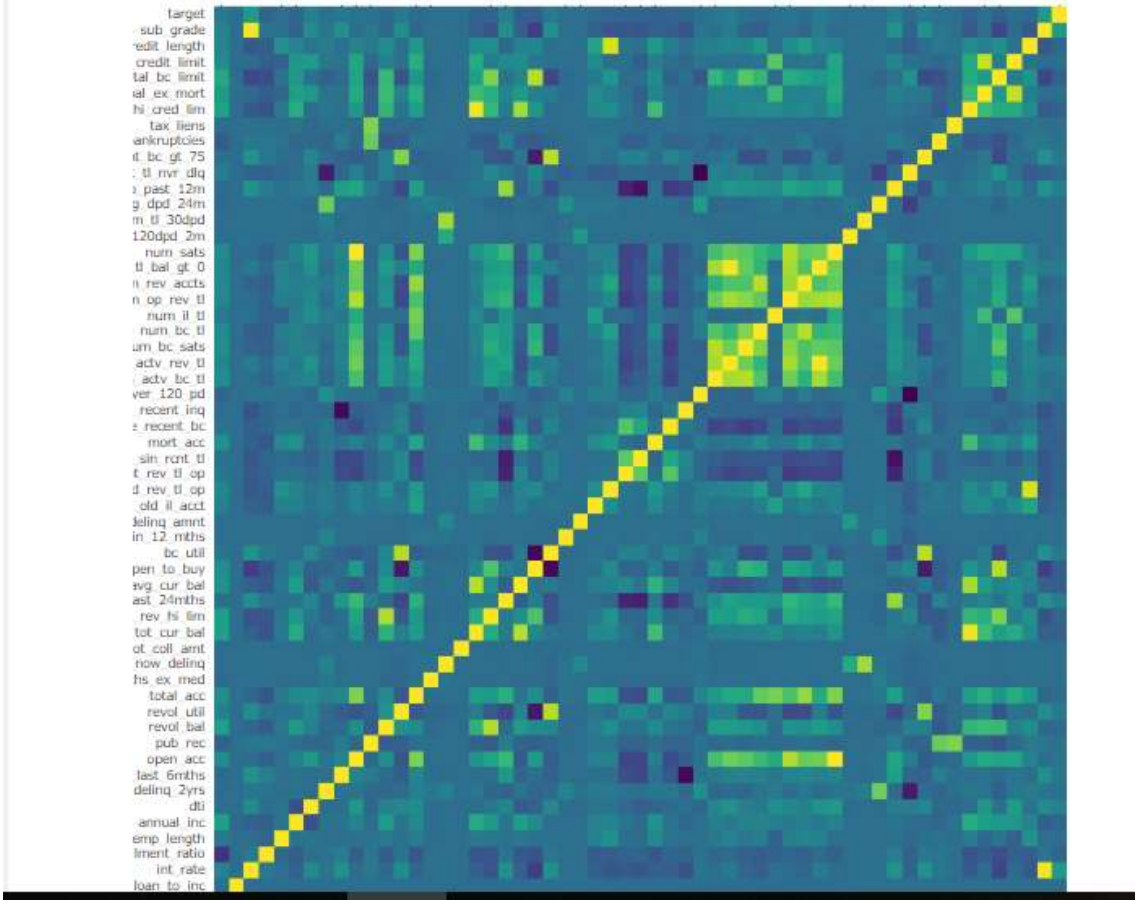


Export to plot.ly »

IV. Correlation Heatmap

In [27]:
```
# calculate the pair-wise correlation
subset = ['loan_amnt', 'loan_to_inc', 'int_rate', 'installment_ratio', 'emp_length',
          'annual_inc', 'dti', 'delinq_2yrs', 'inq_last_6mths', 'open_acc', 'pub_rec',
          'revol_bal', 'revol_util', 'total_acc', 'collections_12_mths_ex_med',
          'acc_now_delinq', 'tot_coll_amt', 'tot_cur_bal', 'total_rev_hi_lim',
          '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_inq',
          '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', 'credit_length', 'sub_grade', 'target']

correlation = train[subset].corr()
```

In [29]:
```
# visualize the pair-wise correlation
annotation_text = np.array([['']] * len(correlation)] * len(correlation))
x = list(correlation.columns)
y = list(correlation.columns)

fig = ff.create_annotated_heatmap(z=correlation.values, x=x, y=y, colorscale='Viridis',
                                  annotation_text=annotation_text)
fig['layout'].update(width=1000, height=1000, margin=go.layout.Margin(t=30))
fig['layout']['xaxis'].update(side='bottom')
iplot(fig)
```



I. H2O Data Preparation

In [4]:
```
# initialize H2O cluster
h2o.init(nthreads=-1, max_mem_size='50G')
h2o.remove_all()
```

```
Checking whether there is an H2O instance running at http://localhost:54321..... not found.
Attempting to start a local H2O server...
  Java Version: openjdk version "1.8.0_171"; OpenJDK Runtime Environment (build 1.8.0_171-8u171-b11-0ubuntu0.16.04.1-b11); OpenJ
DK 64-Bit Server VM (build 25.171-b11, mixed mode)
  Starting server from /home/ubuntu/anaconda3/lib/python3.6/site-packages/h2o/backend/bin/h2o.jar
  Ice root: /tmp/tmpz4g9f12i
  JVM stdout: /tmp/tmpz4g9f12i/h2o_ubuntu_started_from_python.out
  JVM stderr: /tmp/tmpz4g9f12i/h2o_ubuntu_started_from_python.err
  Server is running at http://127.0.0.1:54321
Connecting to H2O server at http://127.0.0.1:54321... successful.
```

H2O cluster uptime:	01 secs
H2O cluster timezone:	Etc/UTC
H2O data parsing timezone:	UTC
H2O cluster version:	3.20.0.1
H2O cluster version age:	1 month and 23 days
H2O cluster name:	H2O_from_python_ubuntu_xk3xwz
H2O cluster total nodes:	1
H2O cluster free memory:	44.44 Gb
H2O cluster total cores:	16
H2O cluster allowed cores:	16
H2O cluster status:	accepting new members, healthy
H2O connection url:	http://127.0.0.1:54321
H2O connection proxy:	None
H2O internal security:	False
H2O API Extensions:	XGBoost, Algos, AutoML, Core V3, Core V4
Python version:	3.6.5 final

In [5]:
```
# transform to H2O Frame, and make sure the target variable is categorical
h2o_train = H2OFrame(train[features + ['target']])
h2o_test = H2OFrame(test[features])

# transform into categorical
h2o_train['target'] = h2o_train['target'].asfactor()

for name in cat_features:
    h2o_train[name] = h2o_train[name].asfactor()
    h2o_test[name] = h2o_test[name].asfactor()
```

```
Parse progress: |████████████████████████████████████████████████| 100%
Parse progress: |████████████████████████████████████████████████| 100%
```

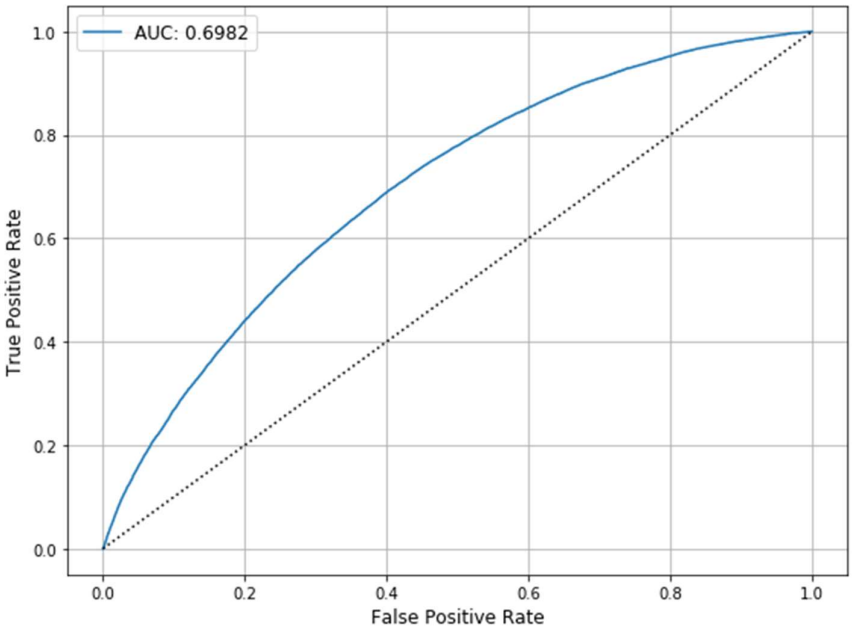
Logistic Regression

In [6]:
```
# create GLM model with 5-folder cross-validation
glm = H2OGeneralizedLinearEstimator(family='binomial', early_stopping=True, nfolds=5,
                                    balance_classes=True, custom_metric_func='auc',
                                    keep_cross_validation_predictions=True, seed=42)

# train logistic regression model using grid search
hyper_parameters = {'alpha': [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
                    'lambda': [0, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]}

# grid search
glm_grid = H2OGridSearch(glm, hyper_parameters)
glm_grid.train(x=features, y='target', training_frame=h2o_train)

# get the grid search result, sorted by AUC decreasing
sorted_glm_grid = glm_grid.get_grid(sort_by='auc', decreasing=True)

best_alpha = sorted_glm_grid.sorted_metric_table()['alpha'][0]
best_lambda = sorted_glm_grid.sorted_metric_table()['lambda'][0]
best_glm_auc = sorted_glm_grid.sorted_metric_table()['auc'][0]

print('Best alpha:\t', best_alpha)
print('Best lambda:\t', best_lambda)
print('Best AUC:\t', best_glm_auc)
```

```
glm Grid Build progress: |████████████████████████████████████████████████| 100%
Best alpha:      [0.6]
Best lambda:     [1.0E-6]
Best AUC:        0.7231989472910549
```

In [7]:
```
# re-build the logistic regression model with best parameters
logit = H2OGeneralizedLinearEstimator(family='binomial', balance_classes=True,
                                      alpha=0.6, lambda_=1.0E-6, seed=42)

logit.train(x=features, y='target', training_frame=h2o_train)

# make prediction
logit_pred = logit.predict(h2o_test).as_data_frame()['p1'].values
```

```
glm Model Build progress: |████████████████████████████████████████████████| 100%
glm prediction progress: |████████████████████████████████████████████████| 100%
```



Random Forest

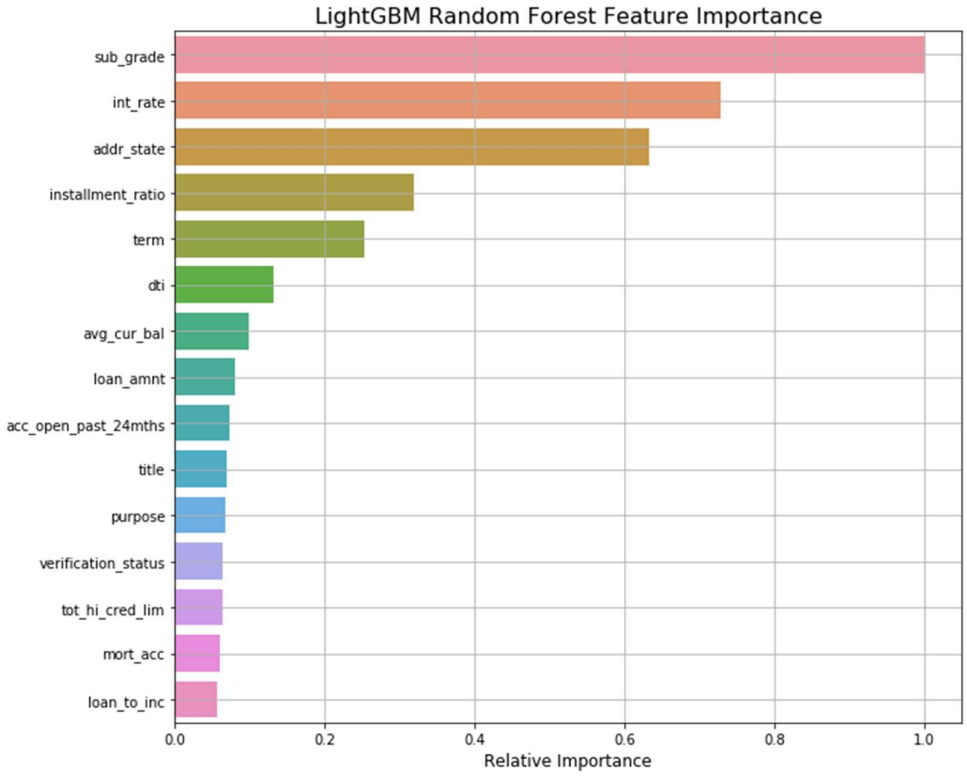
Since it's too slow to run this module, the number of trees `ntrees` are only set to be 100.

In [9]:
```
# random forest grid search
hyper_parameters = {'max_depth': [15, 20, 25, 30, 35],
                    'mtries': [-1, 8, 15, 20, 25],
                    'sample_rate': [0.5, 0.632, 0.75]}

# build random forest model
rf_model = H2ORandomForestEstimator(balance_classes=True, ntrees=100, nfolds=5,
                                    stopping_rounds=5, stopping_metric='auc',
                                    keep_cross_validation_predictions=True)

# define search criteria
search_criteria = {'strategy': 'RandomDiscrete', 'max_models': 10, 'seed': 42,
                   'stopping_metric': 'auc', 'stopping_rounds': 5}

# grid search
rf_grid = H2OGridSearch(rf_model, hyper_parameters, search_criteria=search_criteria)

rf_grid.train(x=features, y='target', training_frame=h2o_train)

# get the grid search result, sorted by AUC decreasing
sorted_rf_grid = rf_grid.get_grid(sort_by='auc', decreasing=True)

best_max_depth = sorted_rf_grid.sorted_metric_table()['max_depth'][0]
best_mtries = sorted_rf_grid.sorted_metric_table()['mtries'][0]
best_sample_rate = sorted_rf_grid.sorted_metric_table()['sample_rate'][0]
best_rf_auc = sorted_rf_grid.sorted_metric_table()['auc'][0]

print('Best max_depth:\t\t', best_max_depth)
print('Best mtries:\t\t', best_mtries)
print('Best sample_rate:\t', best_sample_rate)
print('Best AUC:\t\t', best_rf_auc)
```

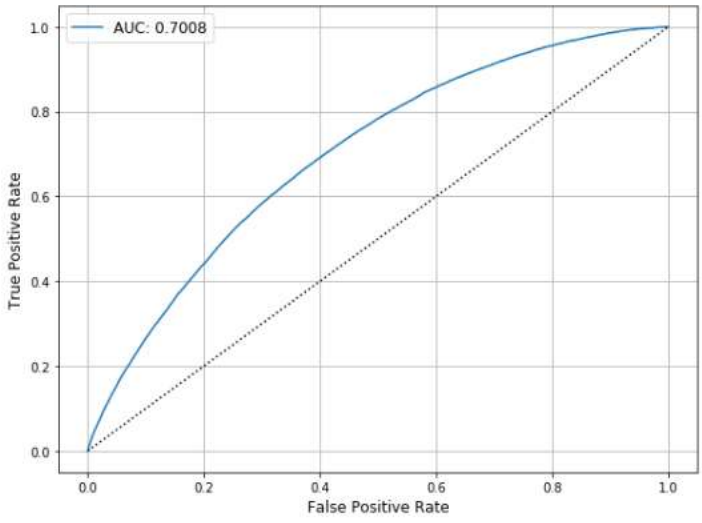
```
drf Grid Build progress: |████████████████████████████████████████| 100%
Best max_depth:          15
Best mtries:             20
Best sample_rate:        0.632
Best AUC:                0.722763705793492
```

In [10]:
```
# build random forest model with best parameters
h2o_rf = H2ORandomForestEstimator(balance_classes=True, max_depth=15, mtries=20,
                                  sample_rate=0.632, ntrees=1000, seed=42)

h2o_rf.train(x=features, y='target', training_frame=h2o_train)

# make prediction
h2o_rf_pred = h2o_rf.predict(h2o_test).as_data_frame()['p1'].values
```

```
drf Model Build progress: |████████████████████████████████████████| 100%
drf prediction progress: |████████████████████████████████████████| 100%
```



In [12]:
```
# build the ROC curve
h2o_rf_fpr, h2o_rf_tpr, _ = roc_curve(test['target'].values, h2o_rf_pred)
h2o_rf_auc = np.round(auc(h2o_rf_fpr, h2o_rf_tpr), 4)
np.save('./result/h2o_rf_fpr_over_sampling.npy', h2o_rf_fpr)
np.save('./result/h2o_rf_tpr_over_sampling.npy', h2o_rf_tpr)

fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(h2o_rf_fpr, h2o_rf_tpr, label='AUC: ' + str(h2o_rf_auc))
ax.plot(h2o_rf_fpr, h2o_rf_fpr, 'k:')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.legend(fontsize=12)
ax.grid(True)
plt.tight_layout()
plt.show()
```



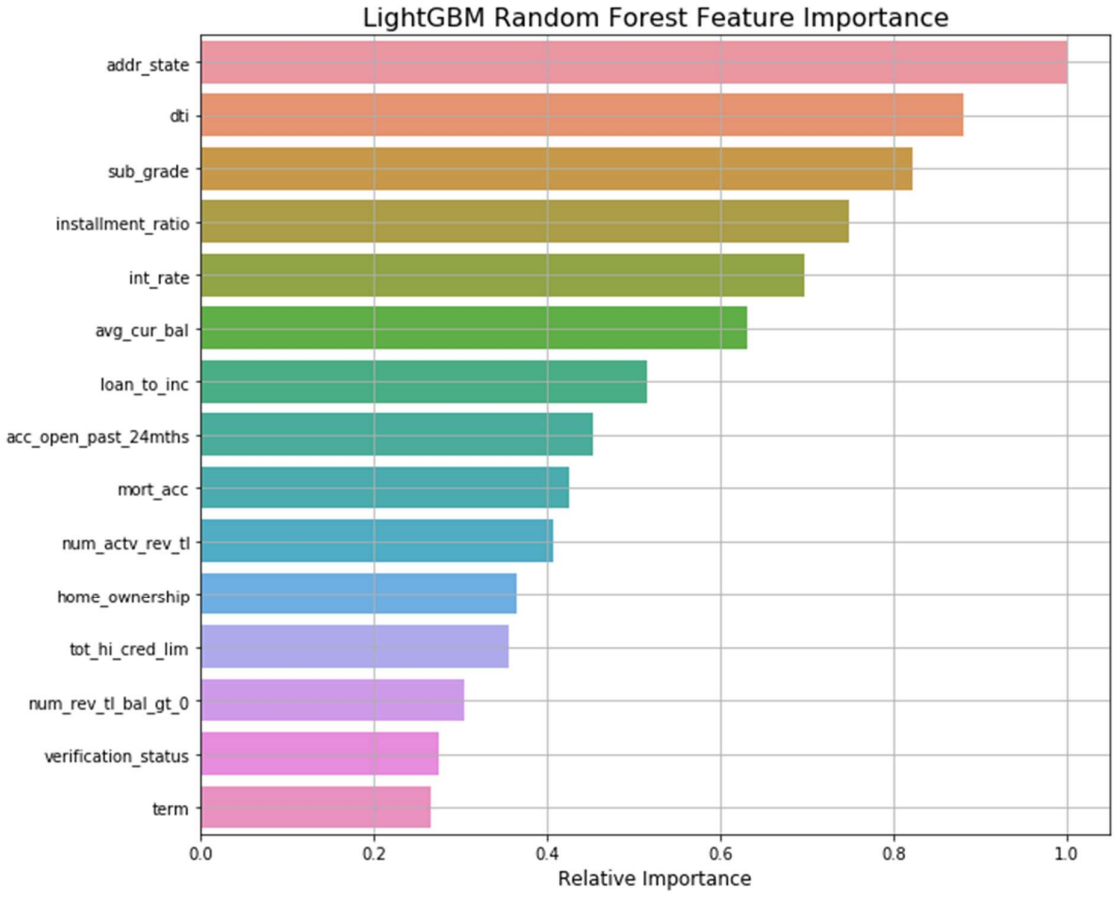
II. LightGBM Data Preparation

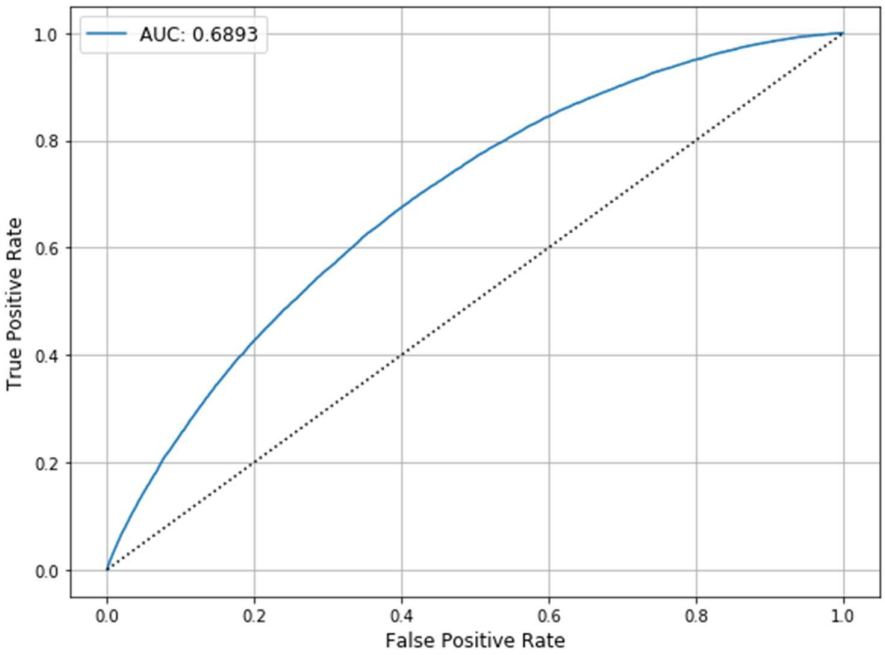
In [14]:
```
# encoding categorical data into numerical format
label_encoders = []
for name in cat_features:
    encoder = LabelEncoder()
    train[name] = encoder.fit_transform(train[name])
    test[name] = encoder.transform(test[name])
    label_encoders.append(encoder)
```

In [15]:
```
# create LightGBM dataset
train_x = train[features]
train_y = train['target'].values

gbm_train = lgb.Dataset(data=train_x, label=train_y, feature_name=features,
                        categorical_feature=cat_features, free_raw_data=False)
```

Random Forest

In [16]:
```
# define parameter space to explore
rf_num_leaves_list = [20, 30, 40, 50, 60]
rf_max_depth_list = [-1, 20, 30, 40, 50]
rf_min_data_in_leaf_list = [20, 30, 40, 50]
rf_bagging_frac_list = [0.5, 0.632, 0.7, 0.8]
rf_feature_frac_list = [0.4, 0.5, 0.6, 0.7, 0.8]

rf_num_leaves_vals = []
rf_max_depth_vals = []
rf_min_data_vals = []
rf_bagging_frac_vals = []
rf_feature_frac_vals = []

rf_mean_auc = []
rf_std_auc = []
```

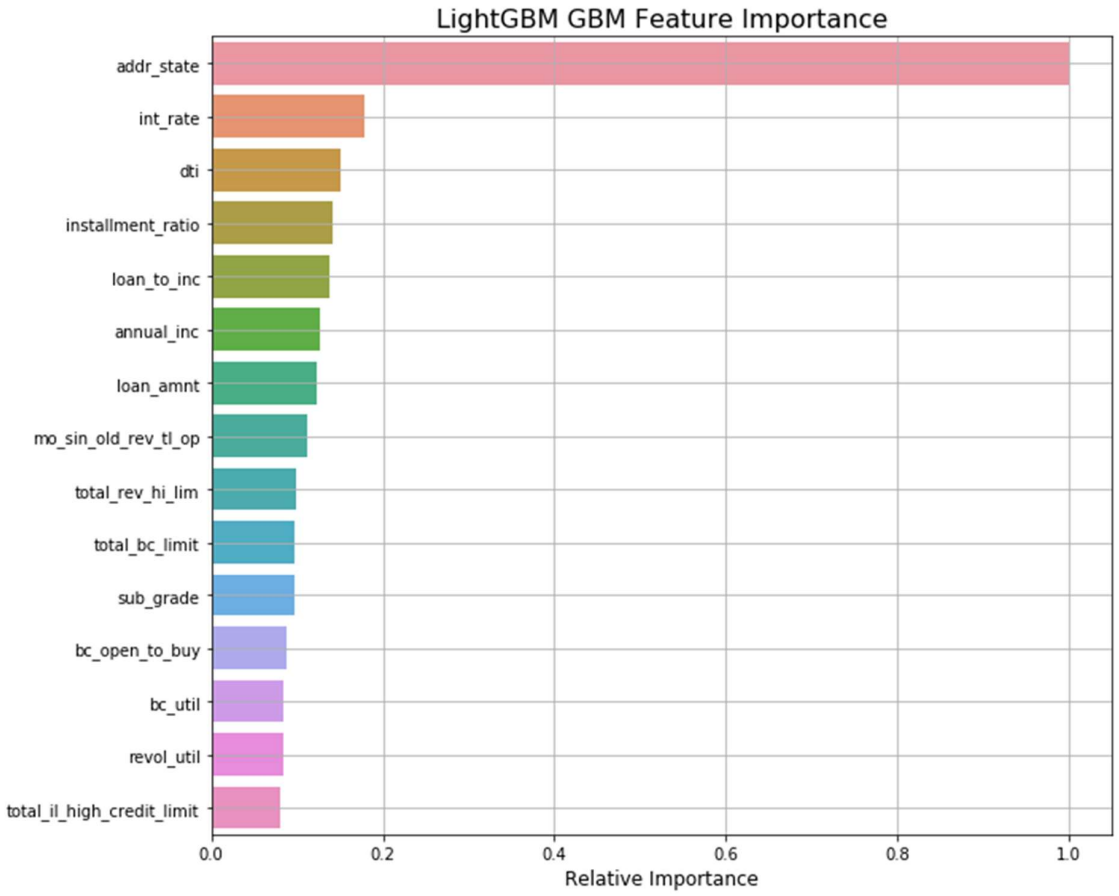


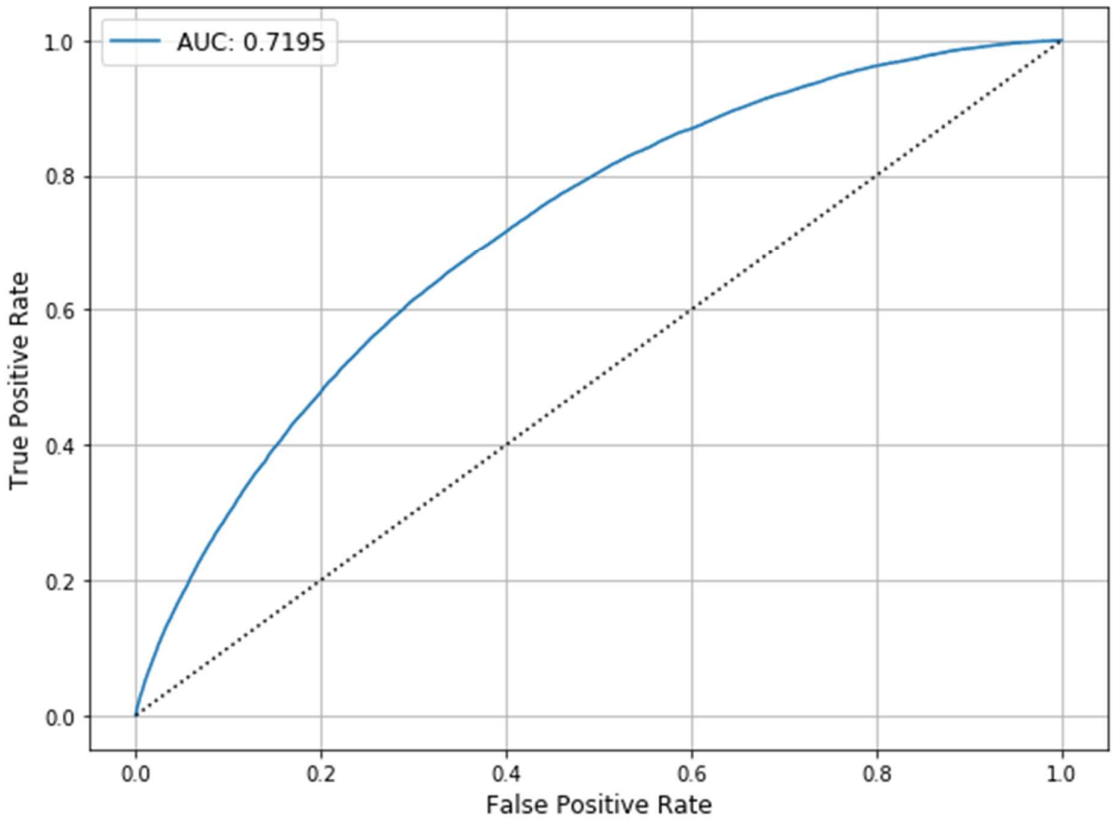


Boosting

In [21]:
```
# define parameter space to explore
gbm_learning_rate_list = [0.03, 0.05, 0.1]
gbm_num_leaves_list = [20, 30, 40, 50, 60]
gbm_max_depth_list = [-1, 10, 20, 30, 40]
gbm_min_data_in_leaf_list = [20, 30, 40, 50]

gbm_learning_rate_vals = []
gbm_num_leaves_vals = []
gbm_max_depth_vals = []
gbm_min_data_vals = []

gbm_best_rounds = []
gbm_mean_auc = []
gbm_std_auc = []
```

In [22]:
```
# Random search with Cross validation
s = '| {0:^3s} | {1:^10s} | {2:^10s} | {3:^16s} | {4:^12s}| {5:^6s} | {6:^6s} |'
print(s.format('learning_rate', 'num_leaves', 'max_depth', 'min_data_in_leaf',
               'best_rounds', 'AUC', 'std'))
print('-' * 94)

# perform random search for given number n
n = 30
np.random.seed(42)
visited = set()
for i in range(n):
    while True:
        learning_rate = np.random.choice(gbm_learning_rate_list)
        num_leaves = np.random.choice(gbm_num_leaves_list)
        max_depth = np.random.choice(gbm_max_depth_list)
        min_data_in_leaf = np.random.choice(gbm_min_data_in_leaf_list)

        tuples = (learning_rate, num_leaves, max_depth, min_data_in_leaf)
        if tuples not in visited:
            visited.add(tuples)
            break

    params = {'objective': 'binary',
              'boosting': 'gbdt',
              'num_threads': 16,
              'is_unbalance': True,
              'metric': ['auc'],
              'max_bin': 255,
              'learning_rate': learning_rate,
              'num_leaves': num_leaves,
              'max_depth': max_depth,
              'min_data_in_leaf': min_data_in_leaf,
              'bagging_fraction': 1.0,
              'feature_fraction': 1.0,
              'bagging_freq': 0,
              'lambda_l1': 0.0,
              'lambda_l2': 0.0,
              'drop_rate': 0.1,
              'seed': 42}
```


LightGBM GBM Feature Importance
addr_state
int_rate
dti
installment_ratio
loan_to_inc
annual_inc
loan_amnt
mo_sin_old_rev_tl_op
total_rev_hi_lim
total_bc_limit
sub_grade
bc_open_to_buy
bc_util
revol_util
total_il_high_credit_limit
0.0
0.2
0.4
0.6
0.8
1.0
Relative Importance


AUC: 0.7195
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

III. CatBoost Data Preparation

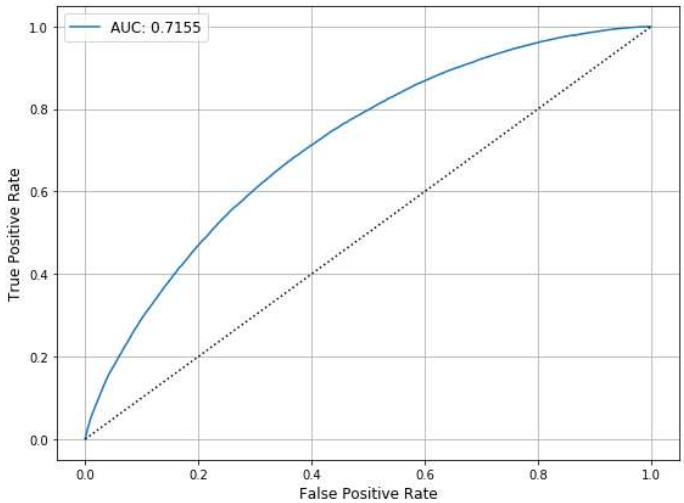
In [26]:
```
# create Pool object
train_pool = Pool(data=train[features], label=train['target'].values, feature_names=features,
                  cat_features=np.array(range(len(cat_features))))

test_pool = Pool(data=test[features], feature_names=features,
                 cat_features=np.array(range(len(cat_features))))
```

Boosting

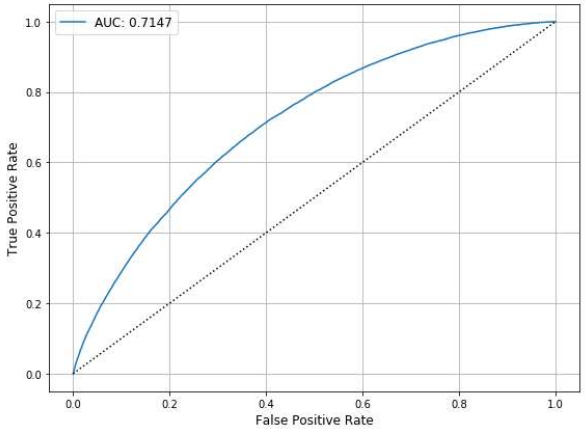
In [27]:
```
# define parameter space to explore
learning_rate_list = [0.03, 0.05, 0.08, 0.1]
depth_list = [4, 5, 6, 7, 8, 9, 10]
l2_leaf_reg_list = [1, 3, 5, 7, 9]
random_strength_list = [0.1, 0.5, 1, 2]
bagging_temperature_list = [0, 0.2, 0.4, 0.6, 0.8, 1.0]

learning_rate_values = []
depth_values = []
l2_leaf_reg_values = []
random_strength_values = []
bagging_temperature_values = []

best_iterations_values = []
train_mean_auc_values = []
test_mean_auc_values = []
```

DataRobot Model Comparison

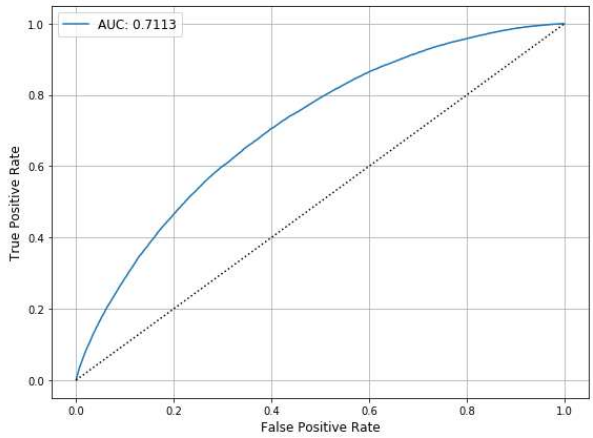
In [15]:
```
# read prediction file
path = './result/H2O_GBM_Classifier_Low_Bias_High_Variance_(39)_80.0_Informative_Features.csv'
pred = pd.read_csv(path)

# calculate the ROC curve
fpr, tpr, _ = roc_curve(test['target'].values, pred['Prediction'])
model_auc = np.round(auc(fpr, tpr), 4)

# visualization
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(fpr, tpr, label='AUC: ' + str(model_auc))
ax.plot(fpr, fpr, 'k:')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.legend(fontsize=12)
ax.grid(True)
plt.tight_layout()
plt.show()
```



In [17]:
```
# read prediction file
path = './result/Light_Gradient_Boosting_on_ElasticNet_Predictions__(49)_80.0_Informative_Features.csv'
pred = pd.read_csv(path)

# calculate the ROC curve
fpr, tpr, _ = roc_curve(test['target'].values, pred['Prediction'])
model_auc = np.round(auc(fpr, tpr), 4)

# visualization
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(fpr, tpr, label='AUC: ' + str(model_auc))
ax.plot(fpr, fpr, 'k:')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.legend(fontsize=12)
ax.grid(True)
plt.tight_layout()
plt.show()
```



In [14]:
```
# read prediction file
path = './result/eXtreme_Gradient_Boosted_Trees_Classifier_with_Ear_(43)_20.0_Informative_Features.csv'
pred = pd.read_csv(path)

# calculate the ROC curve
fpr, tpr, _ = roc_curve(test['target'].values, pred['Prediction'])
model_auc = np.round(auc(fpr, tpr), 4)

# visualization
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(fpr, tpr, label='AUC: ' + str(model_auc))
ax.plot(fpr, fpr, 'k:')
ax.set_xlabel('False Positive Rate', fontsize=12)
ax.set_ylabel('True Positive Rate', fontsize=12)
ax.legend(fontsize=12)
ax.grid(True)
plt.tight_layout()
plt.show()
```



Result and Conclusion

We successfully designed and developed a model for the predicting whether a prospective borrower would “charge-off using random forest, decision tree and boosting algorithm. We were able to predict whether a borrower will fall in the low, medium, high interest rate range. Predicting whether a borrower would receive the complete applied loan amount from the investor was achieved. Our average accuracy was 74.6% whereas the best accuracy achieved was 81.4%.

References

- https://www.kaggle.com/wendykan/lending-club-loan-data
- https://www.lendingclub.com/auth/login?login_url=%2Finfo%2Fdownload-data.action
- https://ieeexplore.ieee.org/document/7803017/
- https://www.researchgate.net/publication/327836640_Peer-to-Peer_Lending_Business_Model_Analysis_and_the_Platform_Dilemma_in_International_Journal_of_Finance_Economics_and_Trade_IJFET_submitted_August_1st_2018_Accepted_Sept_24th
- https://www.semanticscholar.org/paper/Credit-Risk-Analysis-in-Peer-to-Peer-Lending-System-Vinod-Natarajan/c0d182ab22921c2f914770dd83a195c8c2202d6f
- https://www.semanticscholar.org/paper/Credit-Risk-Analysis-in-Peer-to-Peer-Lending-System-Vinod-Natarajan/c0d182ab22921c2f914770dd83a195c8c2202d6f
- http://iosrjournals.org/iosr-jbm/papers/IESMCRC/Volume%201/79.85.pdf
- https://www.adb.org/sites/default/files/publication/478611/adbi-wp912.pdf