In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import locale
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)

locale.setlocale(locale.LC_ALL, 'en_US')
from datetime import datetime
import seaborn as sns

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import locale
from matplotlib.ticker import (MultipleLocator, FormatStrFormatter,
                               AutoMinorLocator)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
locale.setlocale(locale.LC_ALL, 'en_US')
from datetime import datetime
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
```

Chapter1: Exploratory Data Analysis

Read Data

In [2]:
```
data = pd.read_csv('Data/train.csv')
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

        # Rename the columns
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        # Print some summary information
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        # Return the dataframe with missing information
        return mis_val_table_ren_columns
missing_values_table(data)
```

```
Your selected dataframe has 13 columns.
There are 1 columns that have missing values.
```

Out[2]:

	Missing Values	% of Total Values
industry	1	0.0

In [3]:
```
data.head()
```

Out[3]:

	id	industry	state	request_date	term	employee_count	business_new	business_
0	4050975007	Others	VA	27-Apr-10	34	4	New	
1	3735095001	Manufacturing	CA	05-Nov-09	107	1	New	
2	3936555004	Trading	CA	26-Feb-10	84	1	New	
3	4130405000	Engineering	MI	10-Jun-10	240	21	New	
4	4263615008	Education	NH	23-Sep-10	36	1	Existing	

Data Cleaning & Feature Engineering

In [4]:
```
data['insured_amount'] =data['insured_amount'].str.replace(',','')
data['insured_amount'] = data['insured_amount'].str.replace('$','')
data['loan_amount'] = data['loan_amount'].str.replace('$','')
data['loan_amount'] = data['loan_amount'].str.replace(',','')
data['industry'] = data['industry'].fillna('unknown')
data['ones'] = np.ones(data.shape[0])
data['loan_amount'] = data['loan_amount'].astype('float32')
data['insured_amount'] = data['insured_amount'].astype('float32')
```

create month and year features from request_date

In [5]:
```
data['request_date'] = data['request_date'].apply(lambda x: datetime.strptime(x,'
data['month'] = data['request_date'].apply(lambda x: x.month)
data['year'] = data['request_date'].apply(lambda x: x.year)
data['ones'] = np.ones(data.shape[0])
```

Univariate Analysis

The target data is imbalanced but the population of minority class is sufficient at 32 percent. Thus there is no need to use upsampling methods on minority class

In [6]:
```
data['default_status'].value_counts(normalize=True)
```

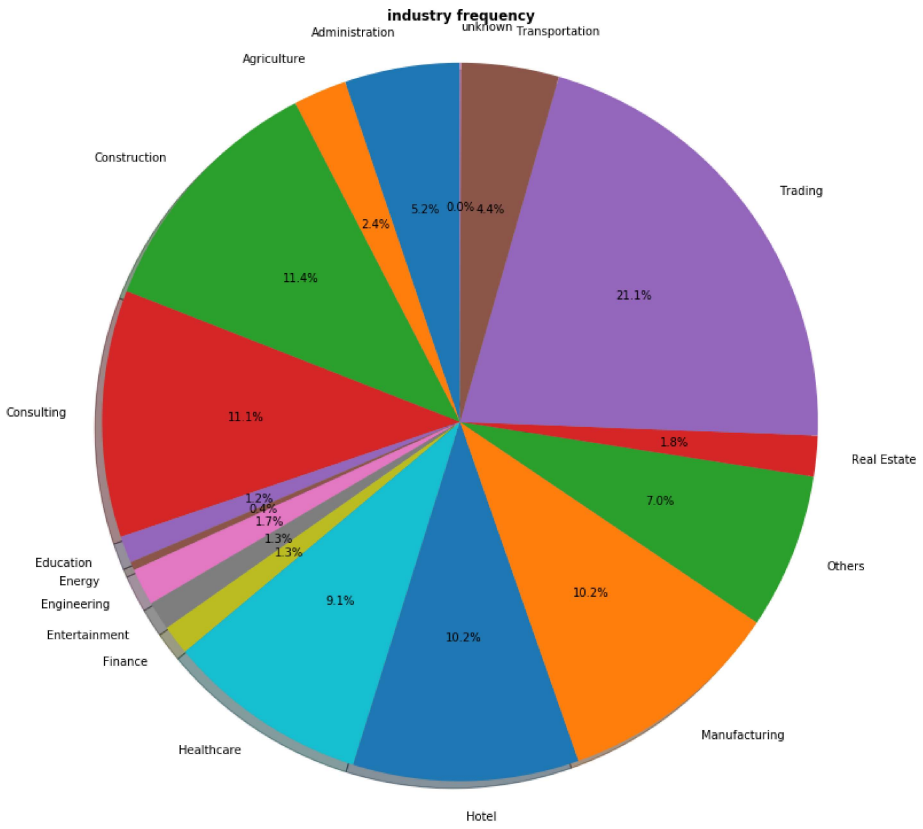
Out[6]:
```
0    0.678185
1    0.321815
Name: default_status, dtype: float64
```

Univariate Analysis:Categorical Variables

The pie charts given below plot the frequency of occurence of different categories for categorical variables

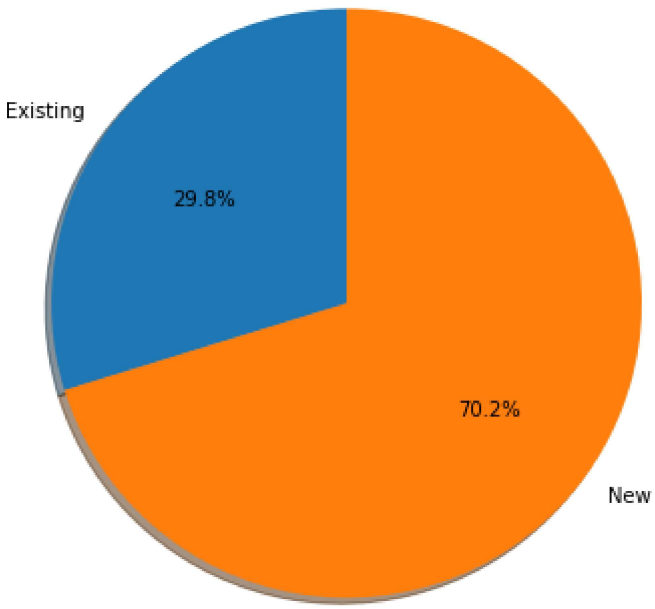
In [7]:
```
def plot_pie_num(data,x,s):
    grouped_data = data.groupby([str(x)]).agg({'ones':'sum'})/data.shape[0]*100
    grouped_data2 = grouped_data.reset_index()
    grouped_data2.rename({'ones':'frequency'},inplace = True, axis = 1)
    #plotting default ratios
    fig1, ax1 =plt.subplots(figsize=s)
    ax1.pie(grouped_data2['frequency'],labels= grouped_data2[str(x)] , autopct='%
        shadow=True, startangle=90)# Equal aspect ratio ensures that pie is drawn
    ax1.axis('equal')
    plt.tight_layout()
    plt.title(str(x)+" frequency",fontweight = "bold")
```

In [8]:
```
plot_pie_num(data,'industry',(15,10))
```

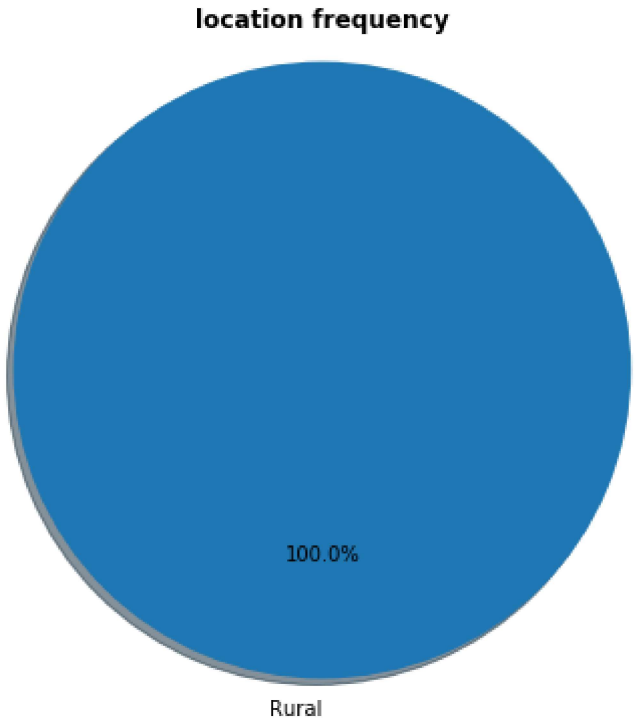


In [9]:
```
plot_pie_num(data,'business_new',(5,10))
print(data['business_new'].value_counts(normalize=True))
```

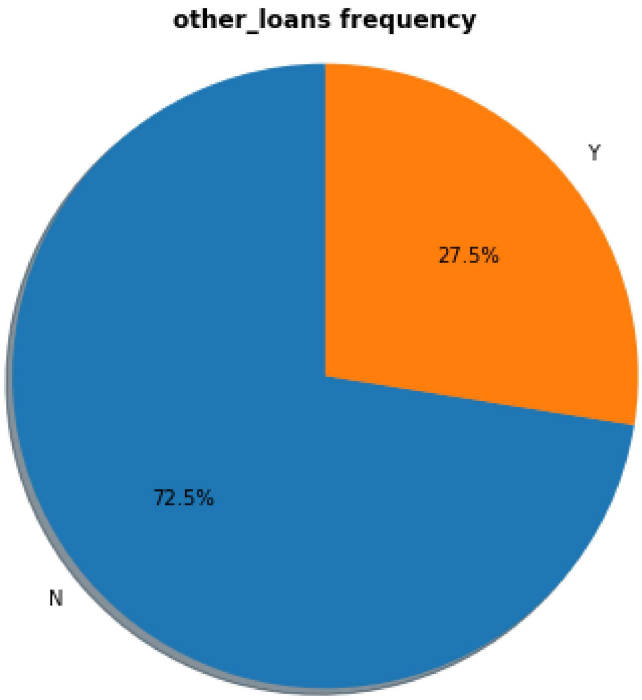
```
New         0.702331
Existing    0.297669
Name: business_new, dtype: float64
```



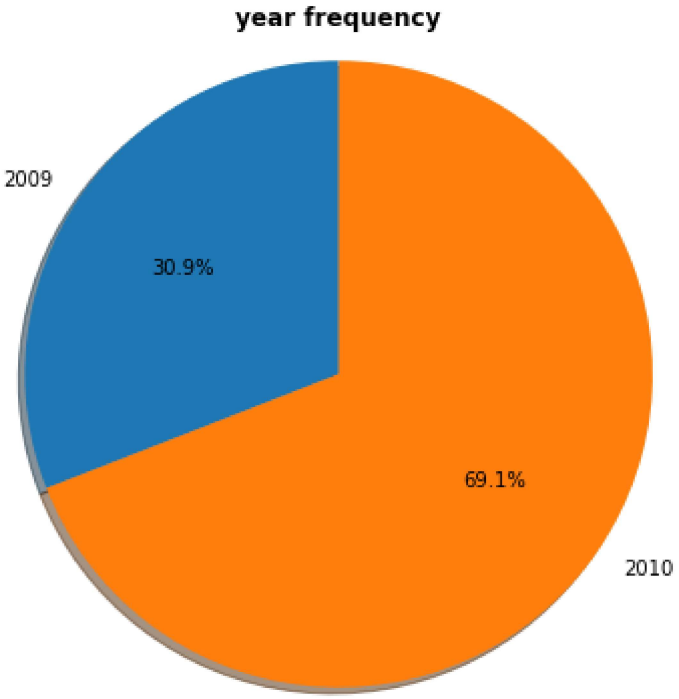
In [10]:
```
plot_pie_num(data,'location',(5,5))
```



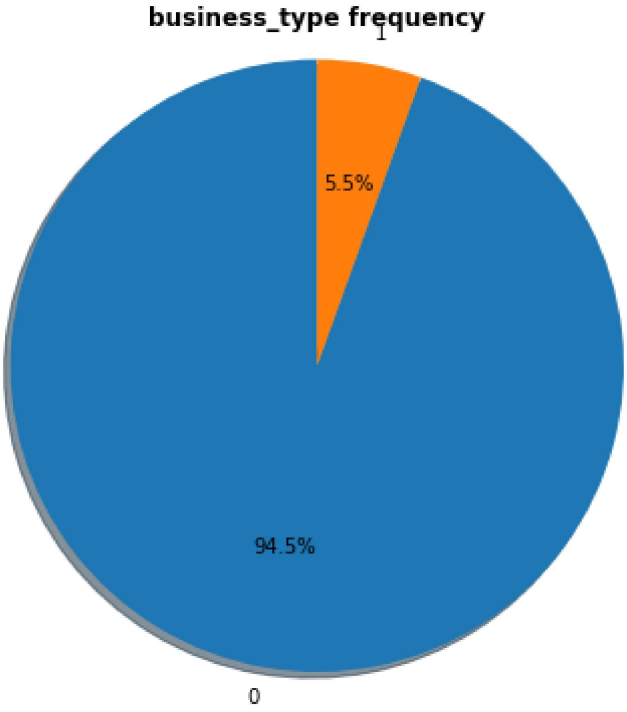
In [11]:
```
plot_pie_num(data,'other_loans',(5,5))
```



In [12]:
```
plot_pie_num(data,'year',(5,5))
```



In [13]:
```
plot_pie_num(data,'business_type',(5,5))
```



Univariate Analysis of Continuous variables

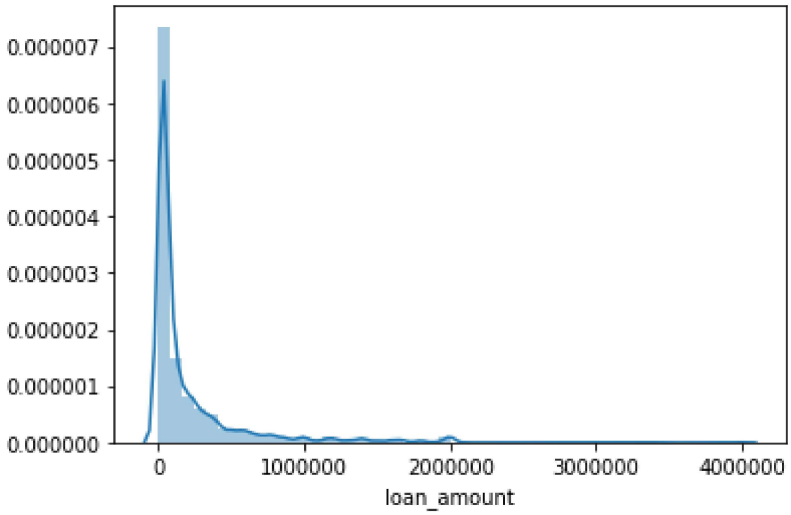
The distributions of the continuous variables are plotted below

loan_amount

Loan amount has a right-skewed distribution as can be seen below

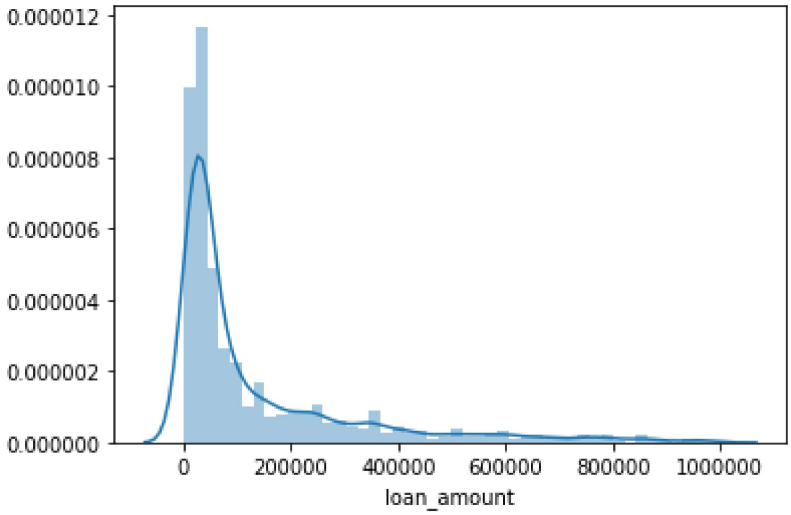
In [14]:
```
sns.distplot(data['loan_amount'])
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f997551dd8>
```



In [15]:
```
sns.distplot(data[data['loan_amount']<1000000.0]['loan_amount'])
```

Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f99756f898>
```

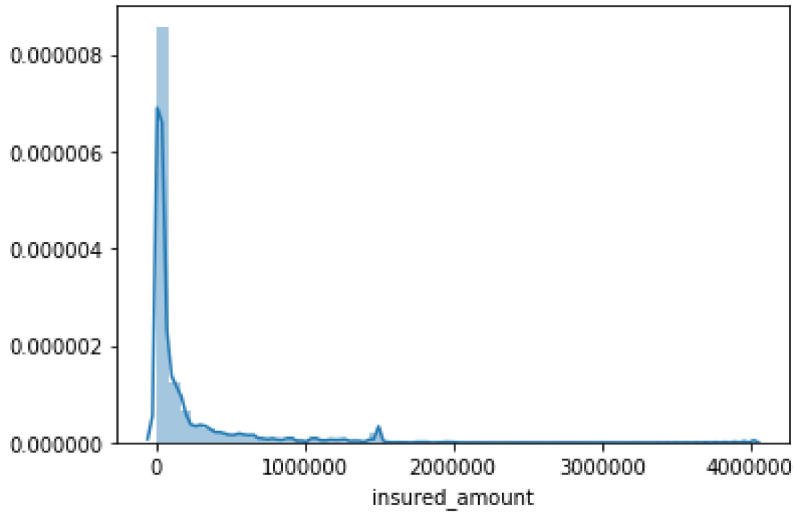


Insured_amount

Insured amount has a right skewed long tailed distribution (poisson/gamma)

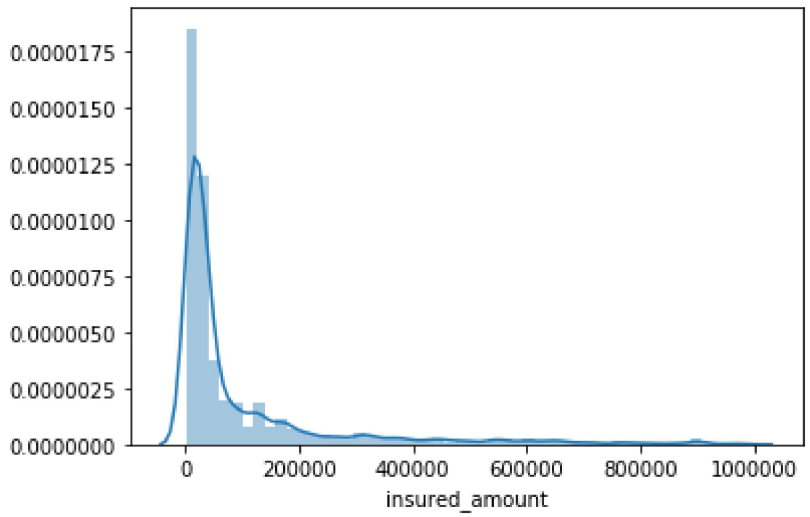
In [16]:
```
sns.distplot(data['insured_amount'])
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f9977b9668>
```



In [17]:
```
sns.distplot(data[data['insured_amount']<1000000.0]['insured_amount'])
```

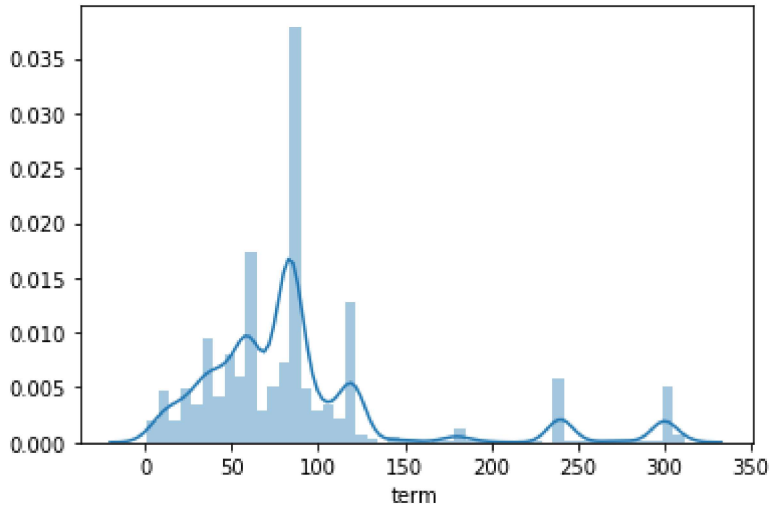
Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f997866898>
```



Term

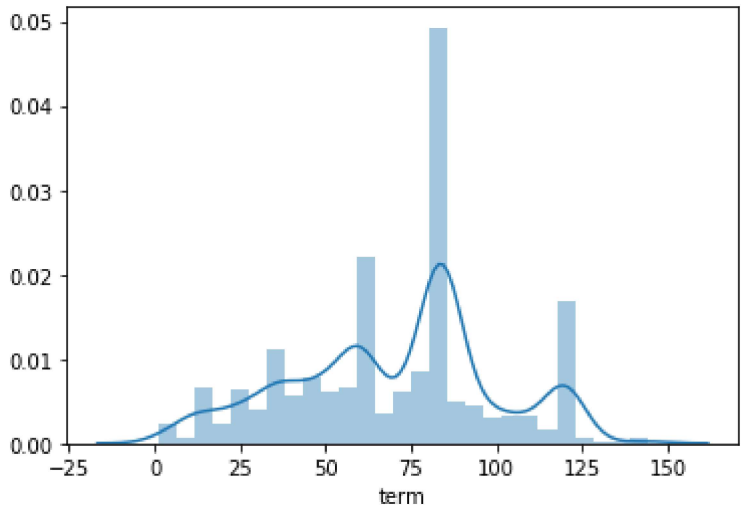
In [18]:
```
sns.distplot(data['term'])
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1f9977ae048>



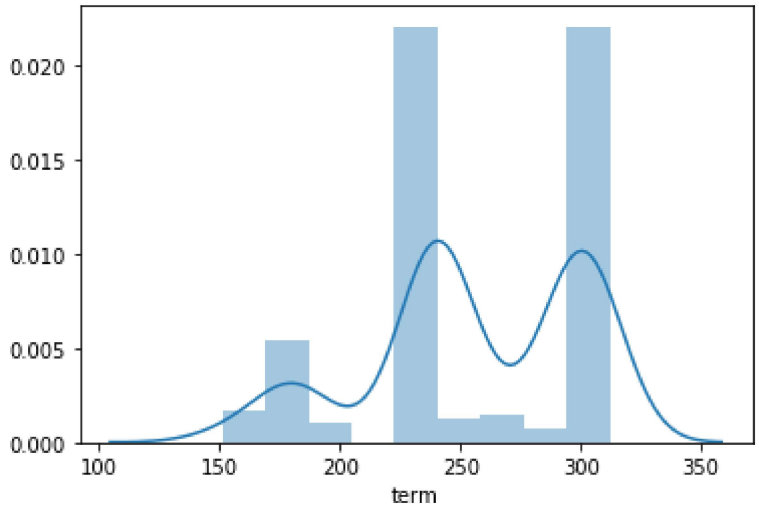
In [19]:
```
sns.distplot(data[data['term']<150]['term'])
```

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x1f9979abcc0>



In [20]:
```
sns.distplot(data[data['term']>150]['term'])
```

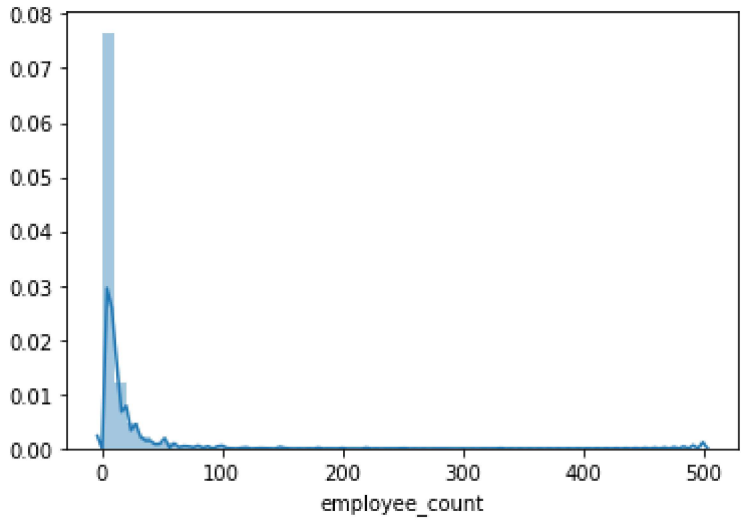
Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f997b779b0>
```



Employee Count

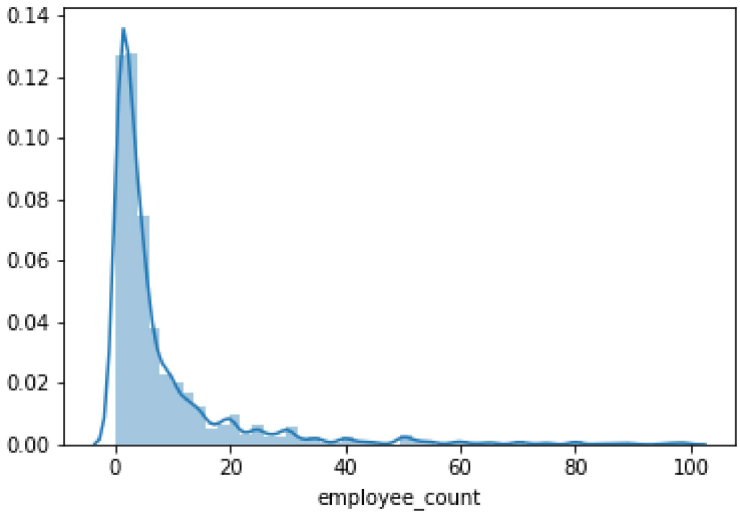
In [21]:
```
sns.distplot(data['employee_count'])
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f997a99b70>
```



In [22]:
```
sns.distplot(data[data['employee_count']<100]['employee_count'])
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1f997ccabe0>
```



Bi-Variate Analysis

In these set of Plots, we try to establish relationship between variables and the target:default_status

In [23]:
```
def plot_hist1(data,x,bins,s,LL,UL):
    #x = 'discount'
    def percen(i):
        return(float("{0:.2f}".format(100*i)))
    bin_var = x+'_binned'
    bin_var2 = x+'_binned2'
    data[bin_var] = pd.cut(data[x], bins,right = False)
    data['ones']=np.ones((data.shape[0]))
    data = data.loc[(data[x]>=LL) & (data[x]<=UL)]
    df_dis = data.groupby([bin_var]).agg({'default_status':'sum','ones':'count'})
    df_dis.rename({'default_status':'defaulted'}, inplace = True,axis = 1)
    df_dis2 = df_dis.reset_index()
    df_dis2[bin_var2] = df_dis2[bin_var].apply(lambda x: str(x))
    df_dis2['not_defaulted'] = df_dis2['ones'] - df_dis2['defaulted']
    df_dis2['avg_default_ratio'] = df_dis2['defaulted']/df_dis2['ones']
    df_dis2['avg_non_default_ratio'] = 1 - (df_dis2['defaulted']/df_dis2['ones'])
    #-------------------------------------------------------------------------
    plt.figure(figsize = s)
    ax0 = plt.axes()
    bars1 = np.array(df_dis2['defaulted'])
    bars2 = np.array(df_dis2['not_defaulted'])
    bars4 = np.hstack((bars1,bars2))
    bars5 = [locale.format("%d", i, grouping=True) for i in bars4]

    height = 0.25*(bins[1] - bins[0])
    midpoints = [i+0.5*height for i in bins][:-1]
    r1 = [i- 0.5*height for i in midpoints ]
    r2 = [i + 0.5*height for i in midpoints]
    r4 = r1 +r2
    labels = np.array(df_dis2[bin_var2])
    plt.barh(r1, bars1,height = height,color = (1,0.5,0.5,1),label ='Number defau
    plt.barh(r2, bars2,height = height,label ='Number not defaulted')
    plt.xlabel('Count',fontsize = 12)
    plt.ylabel(x,fontsize = 12)
    plt.yticks(midpoints,labels)
    plt.title('Count vs '+x)
    a1 =np.linspace(0,np.ceil(np.max(bars4)/100)*100,11)
    plt.xticks(a1,[locale.format("%d", i, grouping=True) for i in a1])
    #plt.yticks(bins)
    for i in range(len(r4)):
        plt.text(y = r4[i]-0.5*height , x = bars4[i]+0.15, s = bars5[i], size = 8
    #plt.tick_params(axis='x', which='minor', bottom=False)
    plt.grid(b=True, which='major', axis='x', color='#666666', linestyle='--', li
    plt.legend()
    plt.show()
    #return df_dis2
#-----------------------------------------------------------------------


    plt.figure(figsize = s)
    ax0 = plt.axes()
    bars2_1 = np.array(df_dis2['avg_default_ratio'])
    bars2_2 = np.array(df_dis2['avg_non_default_ratio'])
    bars2_4 = np.hstack((bars2_1,bars2_2))
    bars2_5 = [percen(i) for i in bars2_4]
    height = 0.25*(bins[1] - bins[0])
    midpoints = [i+0.5*height for i in bins][:-1]
```

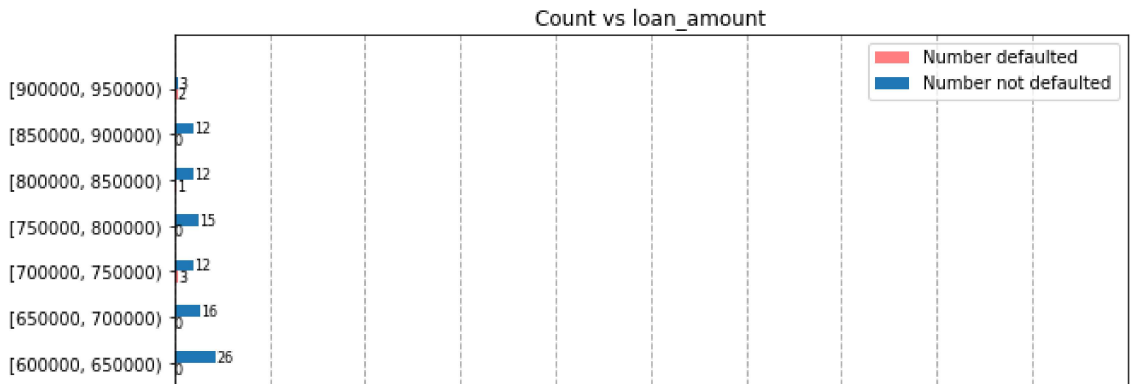
```
    r2_1 = [i- 0.5*height for i in midpoints ]
    r2_2 = [i + 0.5*height for i in midpoints]

    r2_4 = r2_1 +r2_2

    labels = np.array(df_dis2[bin_var2])
    plt.barh(r2_1, bars2_1,height = height,color = (1,0.5,0.5,1),label ='percenta
    plt.barh(r2_2, bars2_2,height = height,label ='percentage not defaulted')
    plt.xlabel('%defaulted/not-defaulted',fontsize = 12)
    plt.ylabel(x,fontsize = 12)
    plt.yticks(midpoints,labels)
    plt.title('percentage vs '+x)
    a1 = np.linspace(0,1,11)
    plt.xticks(a1,[percen(i) for i in a1])
    for i in range(len(r2_4)):
        plt.text(y = r2_4[i]-0.5*height , x = bars2_4[i]+0.005, s = bars2_5[i], s
    #plt.tick_params(axis='x', which='minor', bottom=False)
    plt.grid(b=True, which='major', axis='x', color='#666666', linestyle='--', li
    plt.legend()
    plt.show()
    #return df_dis2
#-----------------------------------------------------------------------

    plt.figure(figsize = s)
    ax0 = plt.axes()
    bars2_1 = np.array(df_dis2['avg_default_ratio'])
    bars2_2 = np.array(df_dis2['avg_non_default_ratio'])
    bars2_4 = np.hstack((bars2_1,bars2_2))
    bars2_5 = [percen(i) for i in bars2_4]
    height = 0.25*(bins[1] - bins[0])
    midpoints = [i+0.5*height for i in bins][:-1]
    r2_1 = [i- 0.5*height for i in midpoints ]
    r2_2 = [i + 0.5*height for i in midpoints]

    r2_4 = r2_1 +r2_2

    labels = np.array(df_dis2[bin_var2])
    plt.barh(r2_1, bars2_1,height = height,color = (1,0.5,0.5,1),label ='percenta
    plt.barh(r2_2, bars2_2,height = height,label ='percentage not defaulted')
    plt.xlabel('%defaulted',fontsize = 12)
    plt.ylabel(x,fontsize = 12)
    plt.yticks(midpoints,labels)
    plt.title('percentage vs '+x)
    a1 = np.linspace(0,1,11)
    plt.xticks(a1,[percen(i) for i in a1])
    for i in range(len(r2_4)):
        plt.text(y = r2_4[i]-0.5*height , x = bars2_4[i]+0.005, s = bars2_5[i], s
    #plt.tick_params(axis='x', which='minor', bottom=False)
    plt.grid(b=True, which='major', axis='x', color='#666666', linestyle='--', li
    plt.legend()
    plt.show()
    #print(df_dis2)
    #return df_dis2
```

Loan_amount vs Default Rate.

we see that the quite counterintutively, the default rates are highest for smaller loans (<50000). The default rates decrease monotonicaly from 50,000 to 600,000. After which the behaviour appears to be random

In [24]:
```
bins = np.arange(0,10**6,5*10**4)
plot_hist1(data,'loan_amount',bins,(10,10),min(bins),max(bins))
#plot_hist1(data,)
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
```

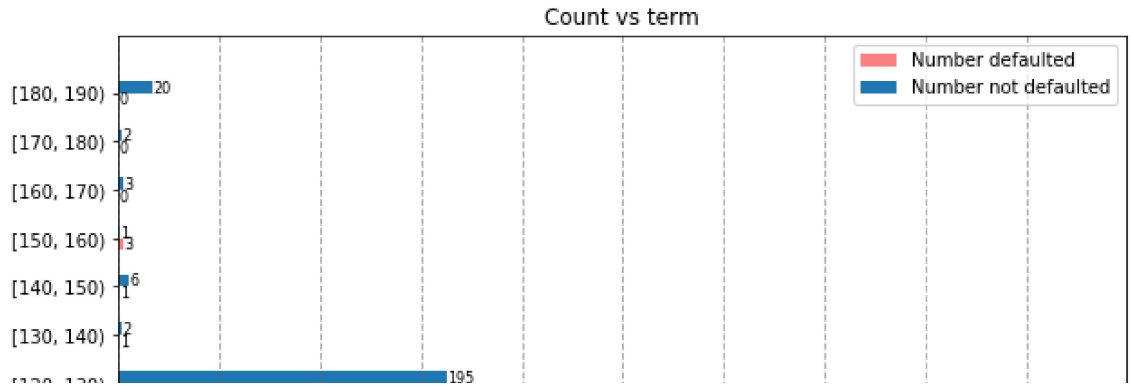


In [25]:
```
data.loc[(data['loan_amount']>=150000) & (data['loan_amount']<200000)]['default_s
```

Out[25]: 18

In [26]:
```
bins = np.arange(0,200,10)
plot_hist1(data,'term',bins,(10,10),min(bins),max(bins))
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
```



Insured Amount vs. Default Rate

The default rates are highest for lower insured amounts. 0to 20000. the Default rates decrease monotonically from 0 to 140000

In [27]:
```
bins = np.arange(0,6*10**5,2*10**4)
plot_hist1(data,'insured_amount',bins,(10,10),min(bins),max(bins))
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: Deprecation
Warning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
```

In [28]:
```
data.loc[(data['insured_amount']>=400000) & (data['insured_amount']<600000)]['de
```

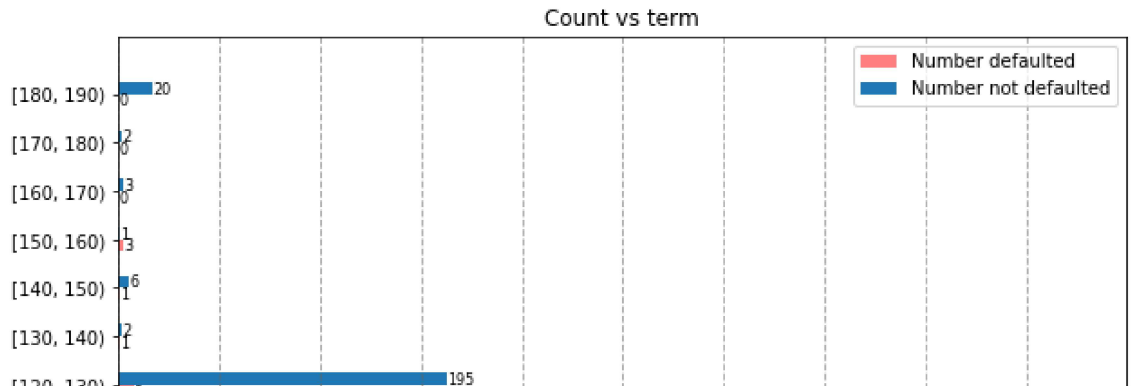
Out[28]: 4

term vs Default Rate

The default rates appear to be sporadically related to loan terms. One cannot draw any conclusive relationship

In [29]:
```
bins = np.arange(0,200,10)
plot_hist1(data,'term',bins,(10,10),min(bins),max(bins))
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Deprecat
ionWarning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: Deprecat
ionWarning: This method will be removed in a future version of Python. Use
'locale.format_string()' instead.
```



In [30]:
```
data.loc[(data['term']>=10) & (data['term']<20)]['default_status'].sum()
```

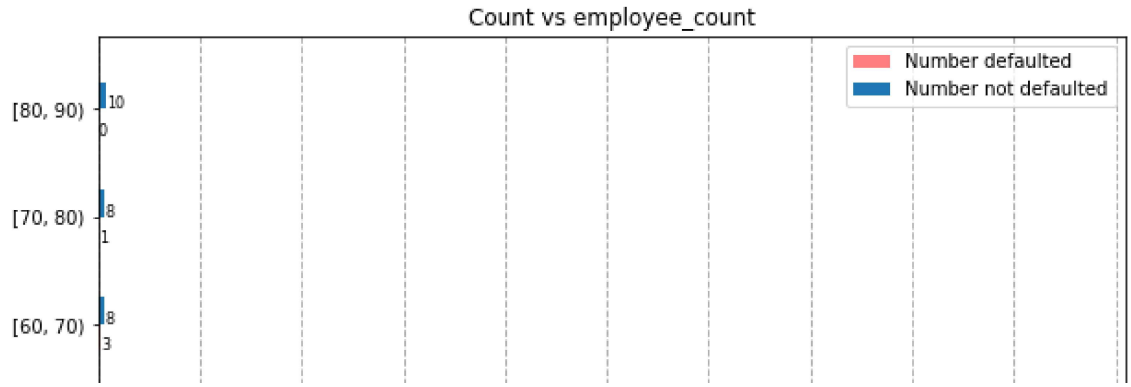
Out[30]: 24

Employee Count Vs. Default Rate

The Default Rate decreases with increas in Employee count. Infact, an employee count of 0-10 people gets the highest default rate

In [31]:
```
bins = np.arange(0,100,10)
plot_hist1(data,'employee_count',bins,(10,10),min(bins),max(bins))
```

```
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:23: Deprecationwarning: This method will be removed in a future version of Python. Use 'locale.format_string()' instead.
C:\Users\ritvik\Anaconda3\lib\site-packages\ipykernel_launcher.py:38: Deprecationwarning: This method will be removed in a future version of Python. Use 'locale.format_string()' instead.
```



In [32]:
```
data.loc[(data['employee_count']>=10) & (data['employee_count']<20)]['default_sta
```

Out[32]: 50

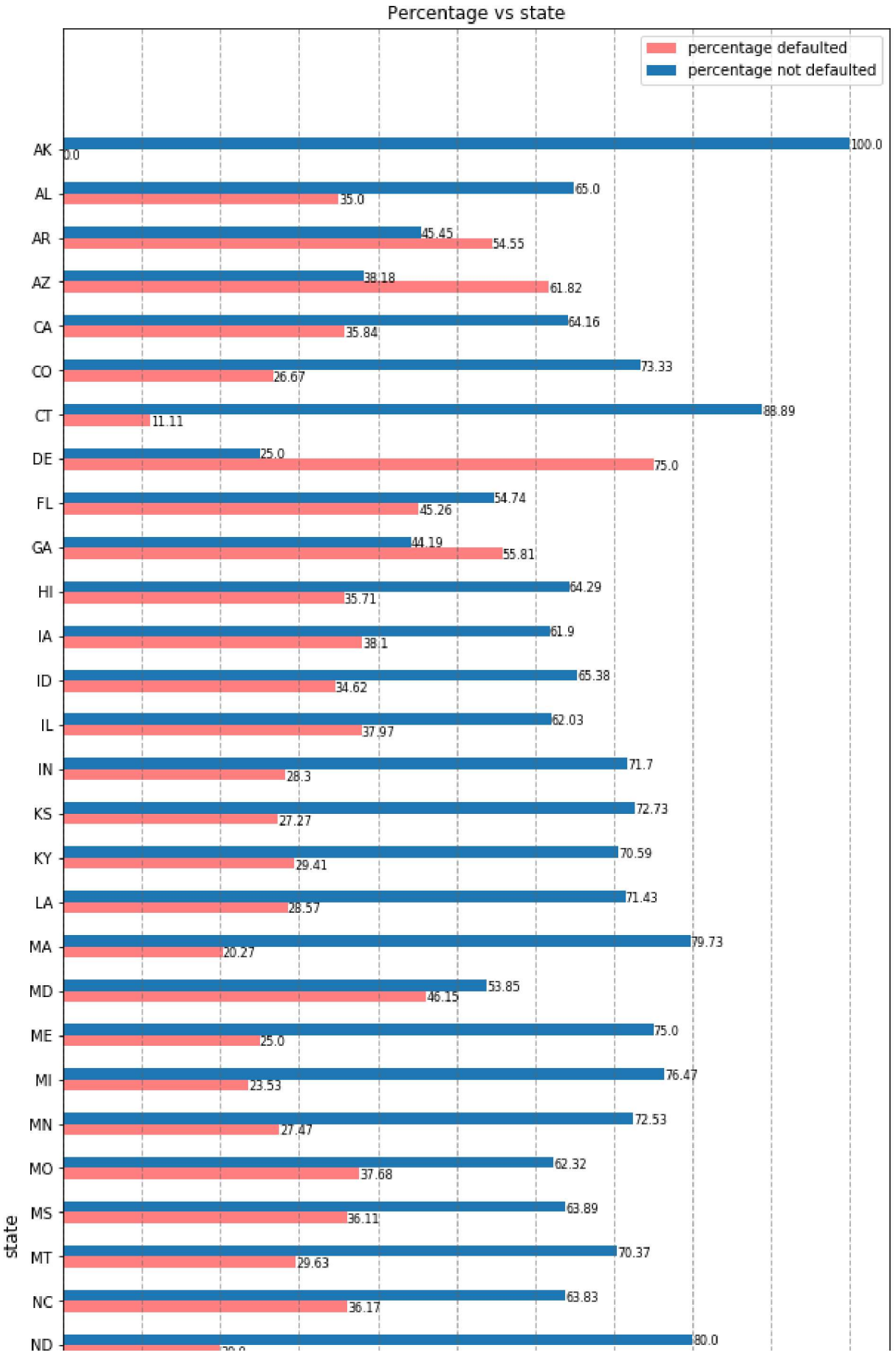
Bi- Variate Analysis of Categorical Variables

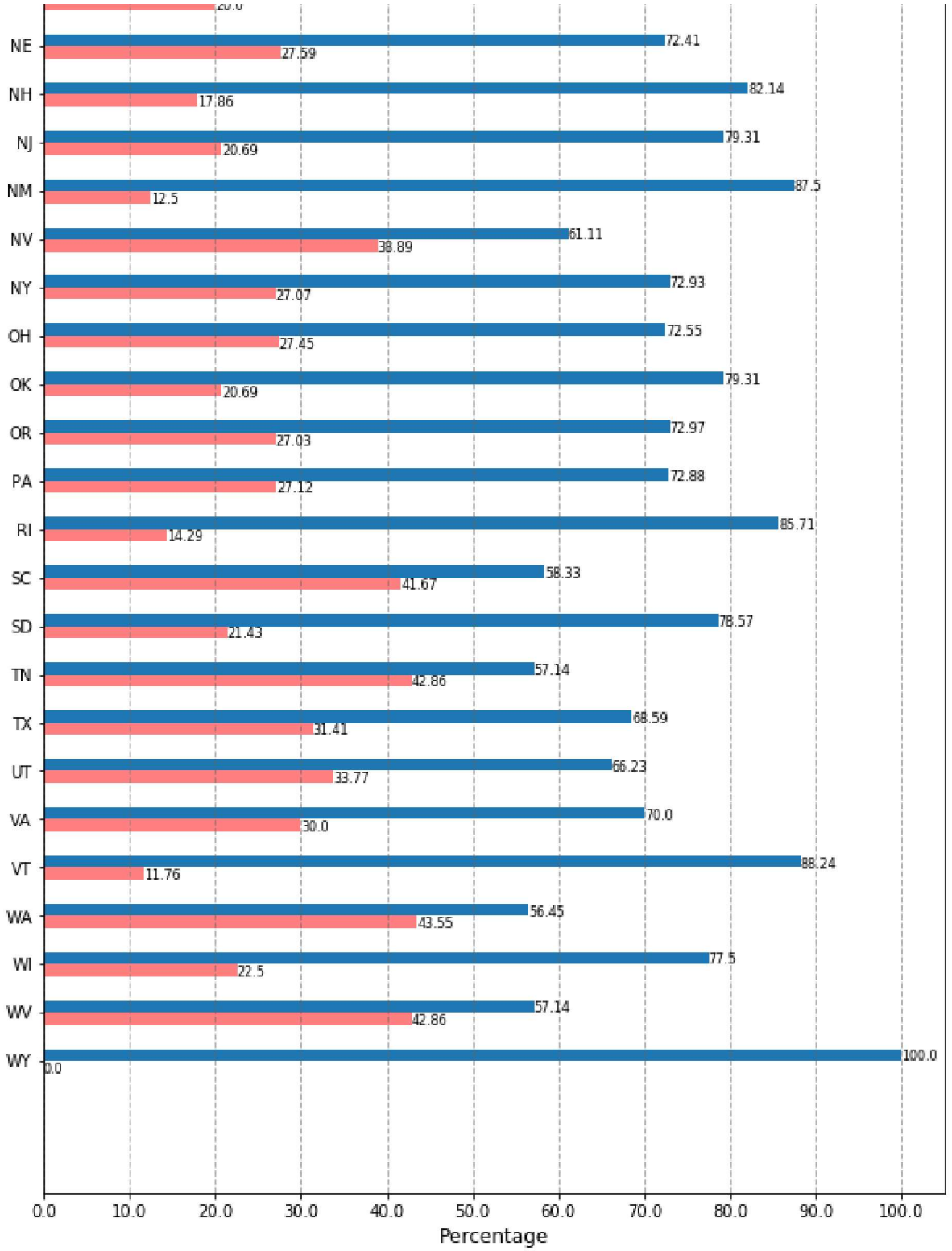
In [33]:
```
#x = 'brand_department'
def plot_discrete_histogram(data,x,bins0,s):
    #x = 'state'
    def percen(i):
        return(float("{0:.2f}".format(100*i)))
    bins = np.linspace(0,100,num =len(bins0))
    data['locator'] = data[x].apply(lambda x:True if x in bins0 else False)
    data = data.loc[(data['locator'])==True]
    data['ones']=np.ones((data.shape[0]))
    df_dis = data.groupby([x]).agg({'default_status':'sum','ones':'count'}).sort_
    df_dis2 = df_dis.reset_index()
    df_dis2['not_defaulted'] = df_dis2['ones'] - df_dis2['default_status']
    df_dis2['avg_default_ratio'] = df_dis2['default_status']/df_dis2['ones']
    df_dis2['avg_non_default_ratio'] = 1 - (df_dis2['default_status']/df_dis2['or
    #-------------------------------------------------------------------------
    #print(df_dis2)
    plt.figure(figsize = s)
    ax0 = plt.axes()
    bars1 = np.array(df_dis2['avg_default_ratio'])
    bars2 = np.array(df_dis2['avg_non_default_ratio'])
    bars4 = np.hstack((bars1,bars2))
    bars5 = [percen(i) for i in bars4]

    height = 0.25*(bins[1] - bins[0])
    midpoints = [i+0.5*height for i in bins]
    r1 = [i- 0.5*height for i in midpoints ]
    #print(len(bins),len(midpoints),len(r1),'*****')
    r2 = [i + 0.5*height for i in midpoints]
    r4 = r1 +r2
    labels = np.array(df_dis2[x])
    #print(len(r1),len(bars1))
    plt.barh(r1, bars1,height = height,color = (1,0.5,0.5,1),label ='percentage d
    #-------------------------------------------------------------------------
    plt.barh(r2, bars2,height = height,label ='percentage not defaulted')
    plt.xlabel('Percentage',fontsize = 12)
    plt.ylabel(x,fontsize = 12)
    plt.yticks(midpoints,labels)
    plt.title('Percentage vs '+x)
    a1 = np.linspace(0,1,11)
    plt.xticks(a1,[percen(i) for i in a1])
    for i in range(len(r4)):
        plt.text(y = r4[i]-0.5*height , x = bars4[i]+0, s = bars5[i], size = 8)
    plt.grid(b=True, which='major', axis='x', color='#666666', linestyle='--', li
    plt.legend()
    plt.show()
```

State vs Default Rate

Delaware appears to have the highest default rate of about 75%

In [34]:
```
bins0 = data['state'].unique()
plot_discrete_histogram(data,'state',bins0,(10,30))
```





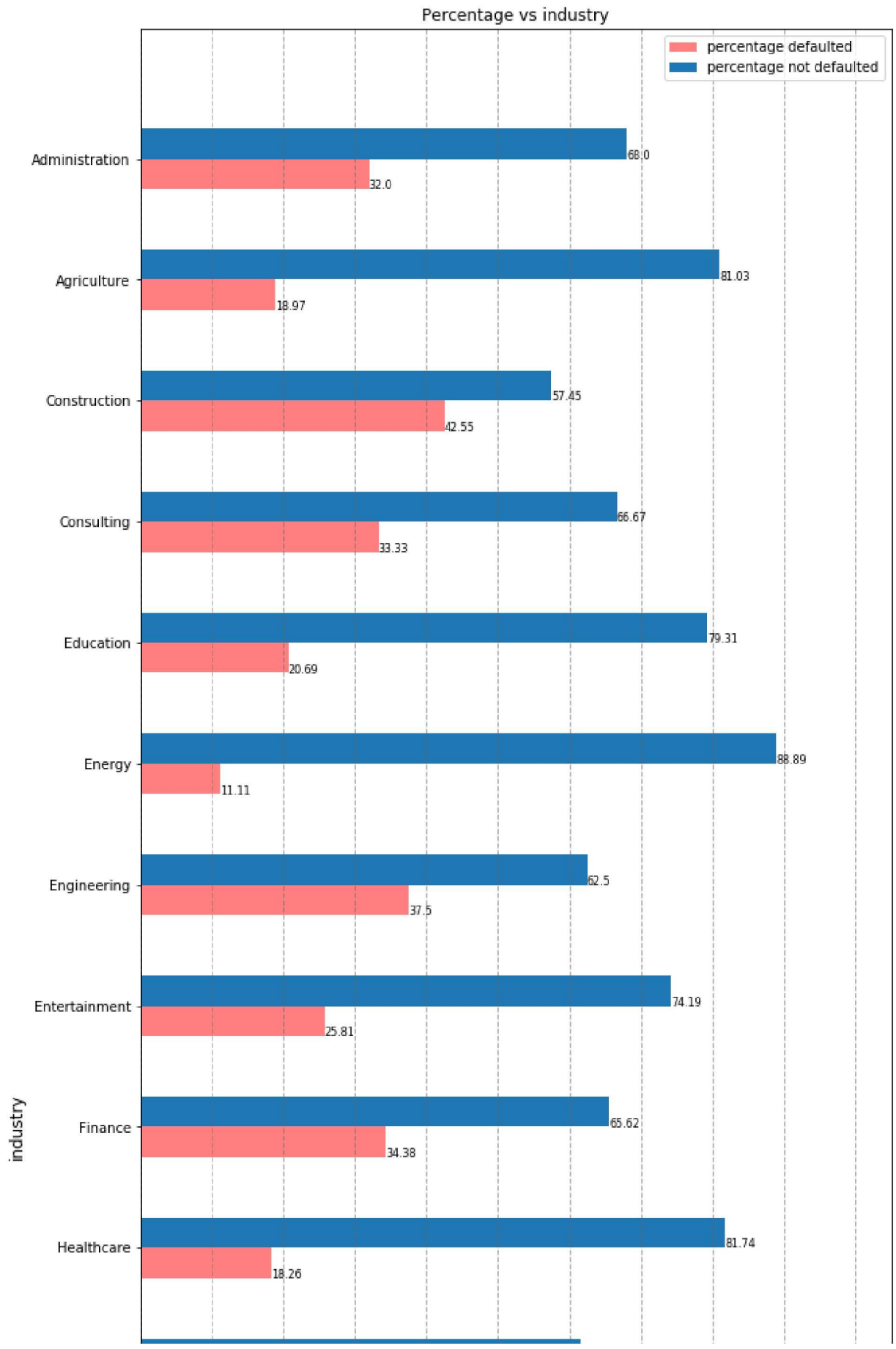
In [35]:
```
data.loc[(data['state']=='MT')]['default_status'].sum()
```

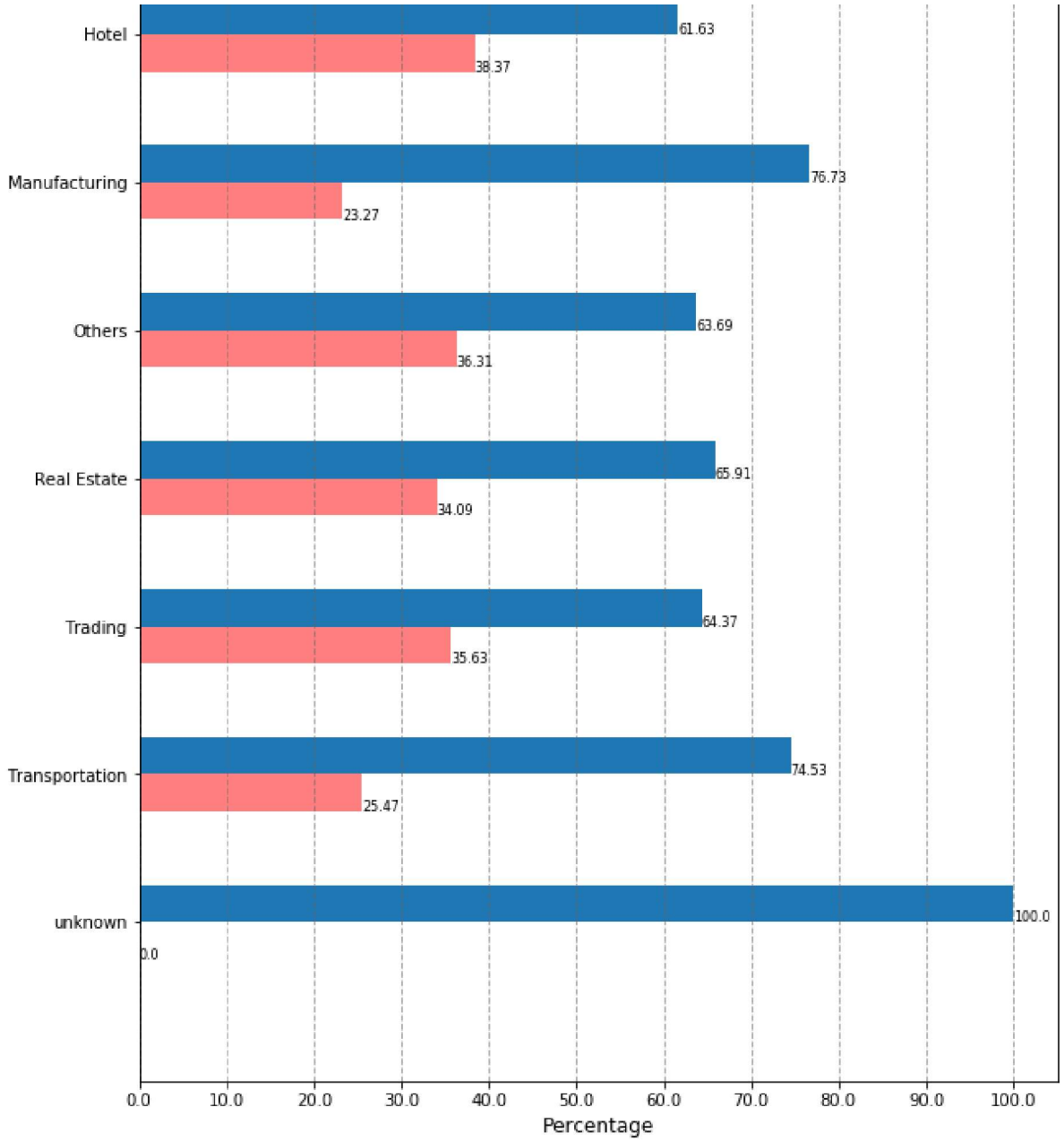
Out[35]: 8

Industry vs Default Rate

Default Rates vary significantly accross industry, with construction and Trading having significantly higher default rates compared to the rest

In [36]:
```
bins0 = data['industry'].unique()
plot_discrete_histogram(data,'industry',bins0,(10,30))
```





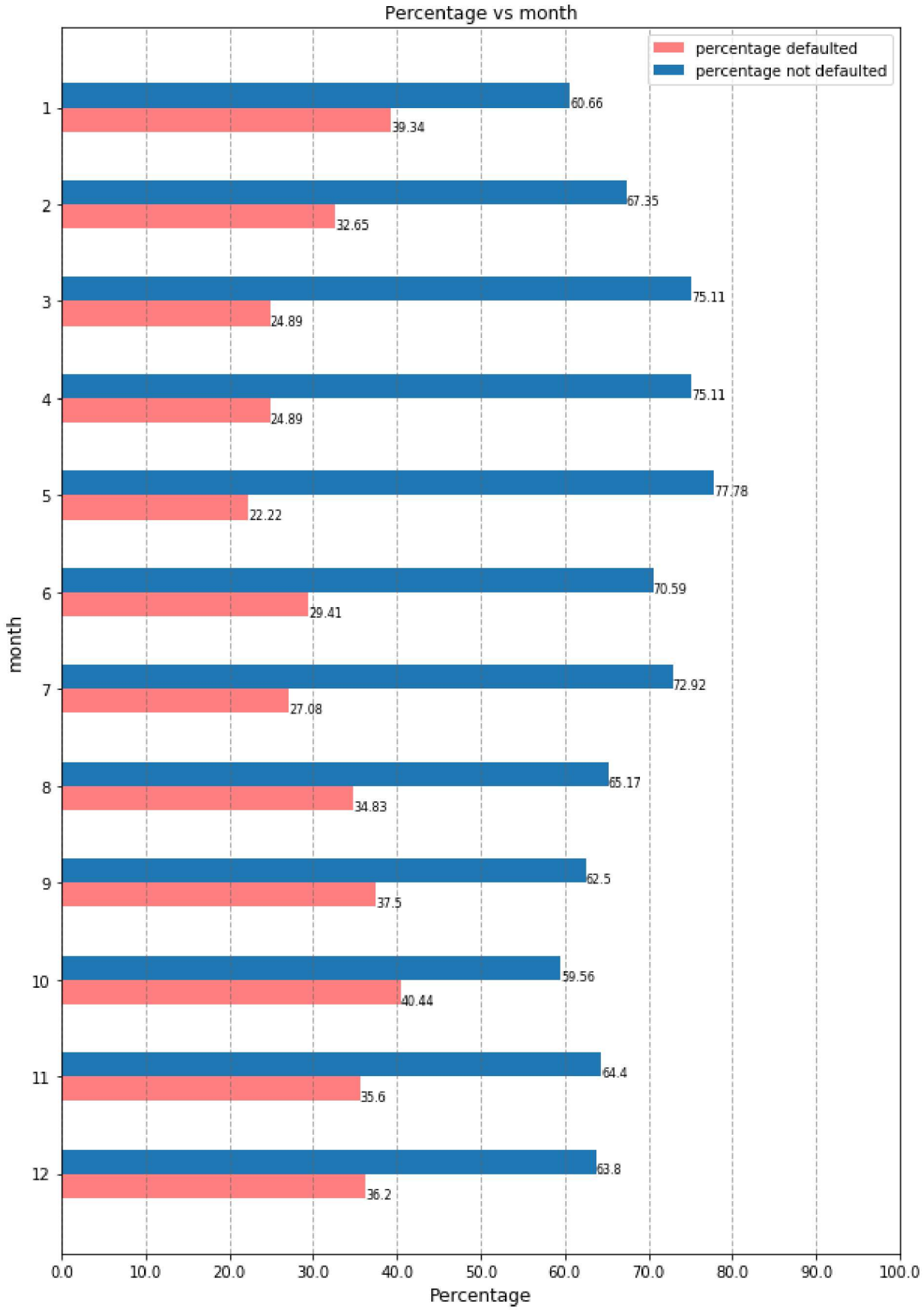
In [37]:
```
data.loc[(data['industry']=='Agriculture')]['default_status'].sum()
```

Out[37]: 11

Month vs Default Rate

Default Rates are highest in January. Does this have anything to do with the Start of the Financial Year?

In [38]:
```
bins0 = data['month'].unique()
plot_discrete_histogram(data,'month',bins0,(10,15))
```



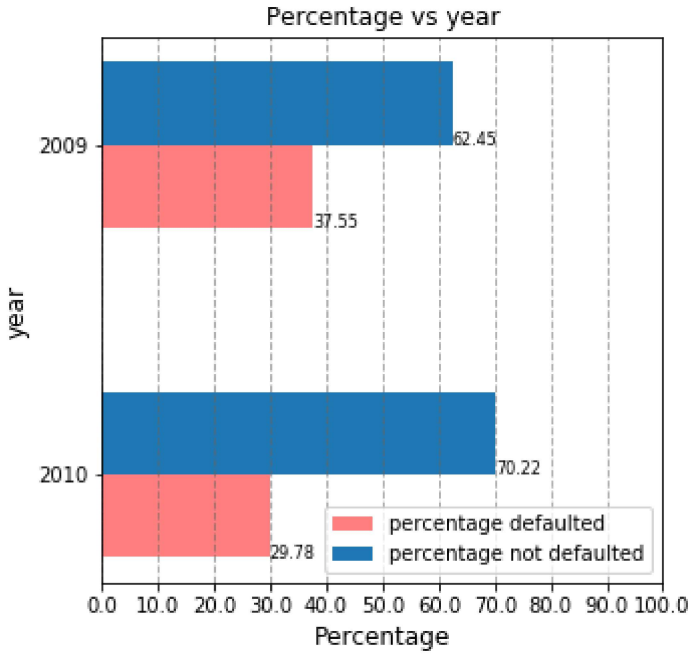
In [39]:
```
data.loc[(data['month']==6)]['default_status'].sum()
```

Out[39]: 40

Year vs. Default Rate

Default Rates are significantly higher in 2009 as compared to 2010. This may be due to the ongoing Global Financial Crisis

In [40]:
```
bins0 = data['year'].unique()
plot_discrete_histogram(data,'year',bins0,(5,5))
```



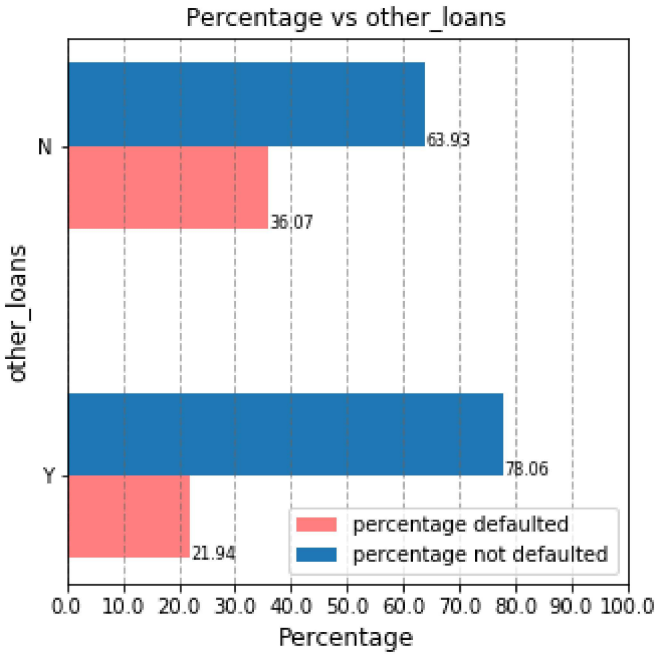
In [41]:
```
data.loc[(data['year']==2009)]['default_status'].sum()
```

Out[41]: 279

Other Loans Vs. Default Rate

Surprisingly, people having other loans have lower default rates than people not having other loans. This may be due to the higher due diligence undertaken on people with more than 1 loan

In [42]:
```
bins0 = data['other_loans'].unique()
plot_discrete_histogram(data,'other_loans',bins0,(5,5))
```



Chapter2: Feature Engineering, Model Training & Tuning

Compute State Default Rate

In [43]:
```
data['ones'] = np.ones(data.shape[0])
data_st = data.groupby(['state']).agg({'default_status':'sum','ones':'sum'})
data_st['state_default_rate'] = data_st['default_status']/data_st['ones']
data_st = data_st.reset_index()[['state','state_default_rate']]
data_st.to_csv('state_default_rate.csv')
data = data.merge(data_st,how = 'left', on = 'state')
```

Compute Industry Default Rate

In [44]:
```
data_id = data.groupby(['industry']).agg({'default_status':'sum','ones':'sum'})
data_id['industry_default_rate'] = data_id['default_status']/data_id['ones']
data_id = data_id.reset_index()[['industry','industry_default_rate']]
data_id.to_csv('industry_default_rate.csv')
data = data.merge(data_id,how = 'left', on = 'industry')
```

Select Features

- The Feature importance for the categorical variable month was seen to be very low. Thus it was dropped.
- There is a very high correlation between loan amount and insured_amount. Thus insured_amount is dropped.

In [45]:
```
data[['loan_amount','insured_amount','employee_count','term','state_default_rate'
```

Out[45]:

	loan_amount	insured_amount	employee_count	term	state_default_rate
loan_amount	1.000000	0.949696	0.236585	0.510284	-0.014164
insured_amount	0.949696	1.000000	0.217003	0.560795	0.002117
employee_count	0.236585	0.217003	1.000000	0.085634	-0.040689
term	0.510284	0.560795	0.085634	1.000000	0.051510
state_default_rate	-0.014164	0.002117	-0.040689	0.051510	1.000000
industry_default_rate	-0.029362	-0.032407	-0.049203	-0.021041	0.021441

In [46]:
```
feature_list = [ \
                 'term',
                 'employee_count',
                 'loan_amount',
                # 'insured_amount',
                 'year',
                 #'month',
                 'state_default_rate',
                 'industry_default_rate',
                 'other_loans',
                 'default_status'
                ]
data2 = data[feature_list]
```

In [47]:
```
data2 = pd.get_dummies(data2,columns=['other_loans',
                                      #  'month',
                                        'year'])

df_y = data2['default_status']
df_x = data2.drop('default_status',axis = 1)
```

Random Forest Hyperparameter Tuning

In [48]:
```
def random_forest_randomsearch(df_x,df_y):
    from sklearn.model_selection import RandomizedSearchCV
    clf = RandomForestClassifier(random_state=42,oob_score = True)
    df_train2_x, df_test2_x, df_train2_y, df_test2_y = train_test_split(df_x, df_
                                                                        test_size

    param_grid =    {
    'n_estimators': range(30,70),
    'max_features': ['sqrt'],
    'max_depth' : range(1,10),
    'min_samples_leaf':range(1,5),
    'min_samples_split':range(2,7),
    'criterion' :['entropy']
    }


    CV_rfc = RandomizedSearchCV(estimator=clf, param_distributions=param_grid, cv
                                n_iter = 20,verbose = 0,scoring = 'accuracy')
    CV_rfc.fit(df_train2_x,df_train2_y)
    results = CV_rfc.cv_results_
    CV_rfc_best_params = CV_rfc.best_params_
    CV_rfc_best_estimator = CV_rfc.best_estimator_
    return CV_rfc_best_estimator,CV_rfc_best_params,results
```

In [49]:
```
import pickle

rf_model,params_rf,results = random_forest_randomsearch(df_x,df_y)
model_name = 'rf_model.sav'
pickle.dump(rf_model, open(model_name, 'wb'))
yhat = rf_model.predict(df_x)
probs = rf_model.predict_proba(df_x)
print('Train accuracy_score',accuracy_score(df_y,yhat), 'Train auc:',roc_auc_scor
print('cross_validation_accuracy_score:',results['mean_test_score'])
```

```
Train accuracy_score 0.9233971690258118 Train auc: 0.978086382251828
cross_validation_accuracy_score: [0.75312314 0.85901249 0.82986318 0.75431291
0.85365854 0.85722784
 0.67519334 0.67519334 0.83105294 0.85841761 0.86555622 0.7584771
 0.81618084 0.85841761 0.84473528 0.75014872 0.84057109 0.75550268
 0.84830458 0.75312314]
```

The Cross validation accuracy scores for the model seem acceptable. The Best Parameters for the Random Forest model are given below-

In [50]:
```
params_rf
```

Out[50]:
```
{'n_estimators': 48,
 'min_samples_split': 3,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 9,
 'criterion': 'entropy'}
```

In [51]:
```
def confusion_matrix1(probs,dataframe_y,n):
    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import f1_score
    thresholds = np.linspace(0.2,0.8,n)
    results= pd.DataFrame()
    counter = 0
    for i in thresholds:
        ypred = np.where(probs>=i,1,0)
        #print(i)
        tn, fp, fn, tp = confusion_matrix(dataframe_y, ypred).ravel()
        results.loc[counter,'threshold'] = i
        results.loc[counter,'fpr'] = fp/(fp + tn)
        results.loc[counter,'FP'] = fp
        results.loc[counter,'tnr'] =tn/(tn+fp)
        results.loc[counter,'TN'] = tn
        results.loc[counter,'fnr'] = fn/(fn+tp)
        results.loc[counter,'FN'] = fn
        results.loc[counter,'tpr'] = tp/(tp+fn)
        results.loc[counter,'TP'] = tp
        results.loc[counter,'acc'] = accuracy_score(dataframe_y,ypred)
        results.loc[counter,'f1_score'] = f1_score(dataframe_y,ypred)
        counter+=1
    return(results)
```

Confusion Matrix

The confusion Matrix is generated for for threshold values from 0.2 to 0.8. We see that the best performance interms of accuracy is achieved around a threshold of 0.5. Thus, the default threshold of 0.5 is retained.

In [52]:
```
confusion_matrix1(probs[:,1],df_y,20)
```

Out[52]:

	threshold	fpr	FP	tnr	TN	fnr	FN	tpr	TP	acc	f1_s
0	0.200000	0.236341	385.0	0.763659	1244.0	0.019405	15.0	0.980595	758.0	0.833472	0.79
1	0.231579	0.192142	313.0	0.807858	1316.0	0.021992	17.0	0.978008	756.0	0.862614	0.82
2	0.263158	0.159607	260.0	0.840393	1369.0	0.024580	19.0	0.975420	754.0	0.883847	0.843
3	0.294737	0.133211	217.0	0.866789	1412.0	0.033635	26.0	0.966365	747.0	0.898834	0.86
4	0.326316	0.105586	172.0	0.894414	1457.0	0.055627	43.0	0.944373	730.0	0.910491	0.87
5	0.357895	0.077962	127.0	0.922038	1502.0	0.071151	55.0	0.928849	718.0	0.924230	0.887
6	0.389474	0.058932	96.0	0.941068	1533.0	0.086675	67.0	0.913325	706.0	0.932140	0.896
7	0.421053	0.046654	76.0	0.953346	1553.0	0.107374	83.0	0.892626	690.0	0.933805	0.896
8	0.452632	0.036219	59.0	0.963781	1570.0	0.128072	99.0	0.871928	674.0	0.934221	0.895
9	0.484211	0.030080	49.0	0.969920	1580.0	0.161708	125.0	0.838292	648.0	0.927560	0.88
10	0.515789	0.025783	42.0	0.974217	1587.0	0.184994	143.0	0.815006	630.0	0.922981	0.87
11	0.547368	0.022099	36.0	0.977901	1593.0	0.216041	167.0	0.783959	606.0	0.915487	0.856
12	0.578947	0.017188	28.0	0.982812	1601.0	0.252264	195.0	0.747736	578.0	0.907161	0.838
13	0.610526	0.012891	21.0	0.987109	1608.0	0.288486	223.0	0.711514	550.0	0.898418	0.818
14	0.642105	0.010436	17.0	0.989564	1612.0	0.331177	256.0	0.668823	517.0	0.886345	0.79
15	0.673684	0.006139	10.0	0.993861	1619.0	0.382924	296.0	0.617076	477.0	0.872606	0.757
16	0.705263	0.003683	6.0	0.996317	1623.0	0.421734	326.0	0.578266	447.0	0.861782	0.729
17	0.736842	0.002455	4.0	0.997545	1625.0	0.470893	364.0	0.529107	409.0	0.846794	0.689
18	0.768421	0.002455	4.0	0.997545	1625.0	0.512290	396.0	0.487710	377.0	0.833472	0.653
19	0.800000	0.001228	2.0	0.998772	1627.0	0.561449	434.0	0.438551	339.0	0.818485	0.608

Display Feature Importances

It is is seen that the term and loan_amount are the most important variables in terms of feature importance.

In [53]:
```
def feature_importances(clf,df_test2_x):
    fimp = clf.feature_importances_
    c1 = df_test2_x.columns.tolist()
    Feature_I = pd.DataFrame(data={'colnames':c1,'Feature_Imp':fimp})
    Feature_I.sort_values('Feature_Imp',ascending =False,inplace=True)
    Feature_I.to_csv('Feature_Importance_May30.csv')
    return (Feature_I)
```

In [54]:
```
feature_importances(rf_model,df_x)
```

Out[54]:

	colnames	Feature_Imp
0	term	0.422931
2	loan_amount	0.269675
3	state_default_rate	0.103593
1	employee_count	0.074234
4	industry_default_rate	0.069793
5	other_loans_N	0.022917
6	other_loans_Y	0.016370
7	year_2009	0.010568
8	year_2010	0.009918

Logistic Regression Model

As can be seen by comparing the accuracy Results for the Sklearn - Logistic Regression Model and the Random Forest Mode, The Random Forest Model is inherently Superior. I will be proceding with the Random Forest model in the future

In [55]:
```
from sklearn.linear_model import LogisticRegressionCV
clf = LogisticRegressionCV(solver= 'lbfgs',cv=5,scoring = 'accuracy')
clf.fit(df_x,df_y)
#clf.scores_
```

Out[55]:
```
LogisticRegressionCV(Cs=10, class_weight=None, cv=5, dual=False,
                     fit_intercept=True, intercept_scaling=1.0, l1_ratios=None,
                     max_iter=100, multi_class='warn', n_jobs=None,
                     penalty='l2', random_state=None, refit=True,
                     scoring='accuracy', solver='lbfgs', tol=0.0001, verbose=0)
```

In [56]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import cross_val_score

predicted = cross_val_predict(LogisticRegression(), df_x, df_y, cv=5)
scores_accuracy = cross_val_score(LogisticRegression(), df_x, df_y, cv=5,scoring=
scores_logloss = cross_val_score(LogisticRegression(), df_x, df_y, cv=5,scoring='
print('accuracy:',scores_accuracy,'neg_log_loss:',scores_logloss)
```

```
accuracy: [0.67775468 0.67775468 0.67775468 0.67916667 0.67849687] neg_log_los
s: [-0.59654738 -0.56500062 -0.55958604 -0.57556998 -0.5487377 ]

C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
```

```
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
C:\Users\ritvik\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:43
2: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

In [57]:
```
#import sklearn
#sorted(sklearn.metrics.SCORERS.keys())
```

Type *Markdown* and LaTeX: α^2

Chapter3: Make Test Predictions

Load Data

In [58]:
```
from datetime import datetime
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
import pickle
data_te = pd.read_csv('Data/test.csv')
data_te['idx'] = range(data_te.shape[0])
```

Feature Engineering

In [59]:
```
data_te['insured_amount'] =data_te['insured_amount'].str.replace(',','')
data_te['insured_amount'] = data_te['insured_amount'].str.replace('$','')
data_te['loan_amount'] = data_te['loan_amount'].str.replace('$','')
data_te['loan_amount'] = data_te['loan_amount'].str.replace(',','')
data_te['industry'] = data_te['industry'].fillna('unknown')
data_te['ones'] = np.ones(data_te.shape[0])
data_te['request_date'] = data_te['request_date'].apply(lambda x: datetime.strpti
data_te['month'] = data_te['request_date'].apply(lambda x: x.month)
data_te['year'] = data_te['request_date'].apply(lambda x: x.year)

df_st = pd.read_csv('state_default_rate.csv')
df_id = pd.read_csv('industry_default_rate.csv')
data_te = data_te.merge(df_st, how = 'left', on = ['state'])
data_te = data_te.merge(df_id, how ='left', on = ['industry'])
```

In [60]:
```
feature_list[0:-1]
```

Out[60]:
```
['term',
 'employee_count',
 'loan_amount',
 'year',
 'state_default_rate',
 'industry_default_rate',
 'other_loans']
```

In [61]:
```
data_te_x = data_te[feature_list[0:-1]]
data_te_x = pd.get_dummies(data_te_x,columns=['other_loans',
                                    #   'month',
                                      'year'])
```

Load Model & Make Prediction

In [62]:
```
rf_model = pickle.load(open('rf_model.sav', 'rb'))
y_pred = rf_model.predict(data_te_x)
y_pred_probs = rf_model.predict_proba(data_te_x)[:,1]
data_te_x['default_status'] = y_pred
data_te_x['idx'] = range(data_te_x.shape[0])
submission = data_te.merge(data_te_x, how = 'inner',on = ['idx'])
submission2=submission[['id','default_status']]
submission2.to_csv('submission_ritvik_dhupkar.csv',index=False)
```

The End ##----------

In [63]:
```
#import sklearn
#sorted(sklearn.metrics.SCORERS.keys())
```

In [ ]:
```

```