In [103]:

```
import pandas as pd
import math
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
import numpy as np
from sklearn.cross_validation import KFold
%matplotlib inline

# Fetching dataset
path=r'/Users/user/Desktop/loans_data.csv'
loan=pd.read_csv(path)
```

In [104]:

```
loan.head()
```

Out[104]:

	ID	Amount.Requested	Amount.Funded.By.Investors	Interest.Rate	Loan.Leng
0	81174.0	20000	20000	8.90%	36 months
1	99592.0	19200	19200	12.12%	36 months
2	80059.0	35000	35000	21.98%	60 months
3	15825.0	10000	9975	9.99%	36 months
4	33182.0	12000	12000	11.71%	36 months

Data Preprocessing

In [105]:

```
#Removing percentage sign from Interest.Rate and Debt.To.Income.Ratio
for colum in ["Interest.Rate","Debt.To.Income.Ratio"]:
    loan[colum]=loan[colum].astype("str")
    loan[colum]=[x.replace("%","") for x in loan[colum]]
```

In [106]:

```
loan.dtypes
```

Out[106]:

```
ID                                float64
Amount.Requested                   object
Amount.Funded.By.Investors         object
Interest.Rate                      object
Loan.Length                        object
Loan.Purpose                       object
Debt.To.Income.Ratio               object
State                              object
Home.Ownership                     object
Monthly.Income                    float64
FICO.Range                         object
Open.CREDIT.Lines                  object
Revolving.CREDIT.Balance           object
Inquiries.in.the.Last.6.Months    float64
Employment.Length                  object
dtype: object
```

We can see that many columns which should have really been numbers have been imported as character columns , probably because some characters values in those columns in the files. We'll convert all such columns to numbers .

In [107]:

```
#Converting columns(which should have been numbers) from character to numbers
for colum in ["Amount.Requested","Amount.Funded.By.Investors","Open.CREDIT.Line
s","Revolving.CREDIT.Balance",
           "Inquiries.in.the.Last.6.Months","Interest.Rate","Debt.To.Income.Rati
o"]:
    loan[colum]=pd.to_numeric(loan[colum],errors="coerce")
```

In [108]:

```
loan.dtypes
```

Out[108]:

```
ID                                float64
Amount.Requested                  float64
Amount.Funded.By.Investors        float64
Interest.Rate                     float64
Loan.Length                        object
Loan.Purpose                       object
Debt.To.Income.Ratio              float64
State                              object
Home.Ownership                     object
Monthly.Income                    float64
FICO.Range                         object
Open.CREDIT.Lines                 float64
Revolving.CREDIT.Balance          float64
Inquiries.in.the.Last.6.Months    float64
Employment.Length                  object
dtype: object
```

In [109]:

```
loan["Loan.Length"].value_counts()
```

Out[109]:

```
36 months    1950
60 months     548
.               1
Name: Loan.Length, dtype: int64
```

In [110]:

```
#Creating dum data frame
dum=pd.get_dummies(loan["Loan.Length"])
```

In [111]:

```
dum.head()
```

Out[111]:

	.	36 months	60 months
0	0	1	0
1	0	1	0
2	0	0	1
3	0	1	0
4	0	1	0

In [112]:

```
#Adding dummy variable for 36 months
loan["month_36"]=dum["36 months"]
```

Now that we'are done with dataframe ll_dummies , we can drop it. Below we demonstrate a general way of removing variables from notebook environment.

In [113]:

```
#Dropping dum data frame
%reset_selective ll_dummies
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])?  y
```

In [114]:

```
who
```

```
KFold    Lasso   LinearRegression        Ridge   colum   dum     dum
my       i       loan
lp_dummy         math    np      path    pd      train_test_split
```

Now that we have created dummies for Loan.Length, we need to remove this from the dataframe.

In [115]:

```
loan=loan.drop('Loan.Length',axis=1)
```

In [116]:

```
loan.dtypes
```

Out[116]:

```
ID                                float64
Amount.Requested                  float64
Amount.Funded.By.Investors        float64
Interest.Rate                     float64
Loan.Purpose                       object
Debt.To.Income.Ratio              float64
State                              object
Home.Ownership                     object
Monthly.Income                    float64
FICO.Range                         object
Open.CREDIT.Lines                 float64
Revolving.CREDIT.Balance          float64
Inquiries.in.the.Last.6.Months    float64
Employment.Length                  object
month_36                            uint8
dtype: object
```

In [117]:

```
loan["Loan.Purpose"].value_counts()
```

Out[117]:

```
debt_consolidation    1307
credit_card            444
other                  200
home_improvement       152
major_purchase         101
small_business          87
car                     50
wedding                 39
medical                 30
moving                  29
vacation                21
house                   20
educational             15
renewable_energy         4
Name: Loan.Purpose, dtype: int64
```

In [118]:

```
round(loan.groupby("Loan.Purpose")["Interest.Rate"].mean())
```

Out[118]:

```
Loan.Purpose
car                    11.0
credit_card            13.0
debt_consolidation     14.0
educational            11.0
home_improvement       12.0
house                  13.0
major_purchase         11.0
medical                12.0
moving                 14.0
other                  13.0
renewable_energy       10.0
small_business         13.0
vacation               12.0
wedding                12.0
Name: Interest.Rate, dtype: float64
```

We can see from the table above that there are 4 effective categoris in the data. Lets club them

In [119]:

```
#Clubbing categories having similar average interest rate
for i in range(len(loan.index)):
    if loan["Loan.Purpose"][i] in ["car","educational","major_purchase"]:
        loan.loc[i,"Loan.Purpose"]="cep"
    if loan["Loan.Purpose"][i] in ["home_improvement","medical","vacation","wedd
ing"]:
        loan.loc[i,"Loan.Purpose"]="hmvg"
    if loan["Loan.Purpose"][i] in ["credit_card","house","other","small_busines
s"]:
        loan.loc[i,"Loan.Purpose"]="chos"
    if loan["Loan.Purpose"][i] in ["debt_consolidation","moving"]:
        loan.loc[i,"Loan.Purpose"]="dg"
```

In [120]:

```
#Making dummies
lp_dummy=pd.get_dummies(loan["Loan.Purpose"],prefix="LP")
```

In [121]:

```
lp_dummy.head()
```

Out[121]:

	LP_cep	LP_chos	LP_dg	LP_hmvg	LP_renewable_energy
0	0	0	1	0	0
1	0	0	1	0	0
2	0	0	1	0	0
3	0	0	1	0	0
4	0	1	0	0	0

In [122]:

```
#Adding it to our original data Dropping Loan.Purpose and LP_renewable_energy. A
lso adding it to our original data
loan=pd.concat([loan,lp_dummy],1)
loan=loan.drop(["Loan.Purpose","LP_renewable_energy"],1)
```

In [123]:

```
loan.dtypes
```

Out[123]:

```
ID                                float64
Amount.Requested                  float64
Amount.Funded.By.Investors        float64
Interest.Rate                     float64
Debt.To.Income.Ratio              float64
State                              object
Home.Ownership                     object
Monthly.Income                    float64
FICO.Range                         object
Open.CREDIT.Lines                 float64
Revolving.CREDIT.Balance          float64
Inquiries.in.the.Last.6.Months    float64
Employment.Length                  object
month_36                            uint8
LP_cep                              uint8
LP_chos                             uint8
LP_dg                               uint8
LP_hmvg                             uint8
dtype: object
```

In [124]:

```
loan["State"].nunique()
```

Out[124]:

```
47
```

In [125]:

```
loan=loan.drop(["State"],1)
```

Next we take care of variable Home.Ownership.

In [126]:

```
loan["Home.Ownership"].value_counts()
```

Out[126]:

```
MORTGAGE    1147
RENT        1146
OWN          200
OTHER          5
NONE           1
Name: Home.Ownership, dtype: int64
```

In [127]:

```
loan["mort"]=np.where(loan["Home.Ownership"]=="MORTGAGE",1,0)
loan["ren"]=np.where(loan["Home.Ownership"]=="RENT",1,0)
loan=loan.drop(["Home.Ownership"],1)
```

In [128]:

```
loan["FICO.Range"].head()
```

Out[128]:

```
0    735-739
1    715-719
2    690-694
3    695-699
4    695-699
Name: FICO.Range, dtype: object
```

In [132]:

```
#Converting to numeric by taking average of range and then dropping FICO.Range.
 p, q
loan['p'], loan['q'] = zip(*loan['FICO.Range'].apply(lambda x: x.split('-', 1)))
```

In [133]:

```
loan["fico"]=0.5*(pd.to_numeric(loan["p"])+pd.to_numeric(loan["q"]))

loan=loan.drop(["FICO.Range","p","q"],1)
```

In [134]:

```
loan["Employment.Length"].value_counts()
```

Out[134]:

```
10+ years    653
< 1 year     249
2 years      243
3 years      235
5 years      202
4 years      191
1 year       177
6 years      163
7 years      127
8 years      108
n/a           77
9 years       72
.              2
Name: Employment.Length, dtype: int64
```

In [136]:

```
loan["Employment.Length"]=loan["Employment.Length"].astype("str")
loan["Employment.Length"]=[x.replace("years","") for x in loan["Employment.Lengt
h"]]
loan["Employment.Length"]=[x.replace("year","") for x in loan["Employment.Lengt
h"]]
```

In [137]:

```
round(loan.groupby("Employment.Length")["Interest.Rate"].mean(),2)
```

Out[137]:

```
Employment.Length
.        11.34
1        12.49
10+      13.34
2        12.87
3        12.77
4        13.14
5        13.40
6        13.29
7        13.10
8        13.01
9        13.15
< 1      12.86
n/a      12.85
nan       7.51
Name: Interest.Rate, dtype: float64
```

In [138]:

```
loan["Employment.Length"]=[x.replace("n/a","< 1") for x in loan["Employment.Leng
th"]]
loan["Employment.Length"]=[x.replace("10+","10") for x in loan["Employment.Lengt
h"]]
loan["Employment.Length"]=[x.replace("< 1","0") for x in loan["Employment.Lengt
h"]]
loan["Employment.Length"]=pd.to_numeric(loan["Employment.Length"],errors="coerc
e")
```

In [139]:

```
loan.dtypes
```

Out[139]:

```
ID                                 float64
Amount.Requested                   float64
Amount.Funded.By.Investors         float64
Interest.Rate                      float64
Debt.To.Income.Ratio               float64
Monthly.Income                     float64
Open.CREDIT.Lines                  float64
Revolving.CREDIT.Balance           float64
Inquiries.in.the.Last.6.Months     float64
Employment.Length                  float64
month_36                             uint8
LP_cep                               uint8
LP_chos                              uint8
LP_dg                                uint8
LP_hmvg                              uint8
mort                                 int64
ren                                  int64
fico                               float64
dtype: object
```

In [140]:

```
loan.shape
```

Out[140]:

```
(2500, 18)
```

In [141]:

```
loan.dropna(axis=0,inplace=True)
```

In [142]:

```
loan.shape
```

Out[142]:

```
(2471, 18)
```

Regular

In [143]:

```
loan_train, loan_test = train_test_split(loan, test_size = 0.2,random_state=2)
```

In [144]:

```
lr=LinearRegression()
```

In [145]:

```
x_train=loan_train.drop(["Interest.Rate","ID","Amount.Funded.By.Investors"],1)
y_train=loan_train["Interest.Rate"]
x_test=loan_test.drop(["Interest.Rate","ID","Amount.Funded.By.Investors"],1)
y_test=loan_test["Interest.Rate"]
```

In [146]:

```
lr.fit(x_train,y_train)
```

Out[146]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normaliz
e=False)
```

In [147]:

```
tst=lr.predict(x_test)

residual=tst-y_test

rmse_lr=np.sqrt(np.dot(residual,residual)/len(tst))

rmse_lr
```

Out[147]:

```
1.9984182813784865
```

In [149]:

```
#Getting coefficients
coef=lr.coef_

feature=x_train.columns

list(zip(feature,coef))
```

Out[149]:

```
[('Amount.Requested', 0.0001647141651302124),
 ('Debt.To.Income.Ratio', 0.0019407167638444051),
 ('Monthly.Income', -1.9644954030405461e-05),
 ('Open.CREDIT.Lines', -0.034083616785865634),
 ('Revolving.CREDIT.Balance', -3.9668091912914427e-06),
 ('Inquiries.in.the.Last.6.Months', 0.35395352269202873),
 ('Employment.Length', 0.0062596138442057025),
 ('month_36', -3.1338448528798915),
 ('LP_cep', -0.36782330890011328),
 ('LP_chos', -0.24412655191507573),
 ('LP_dg', -0.43656408581180073),
 ('LP_hmvg', -0.44251741243247306),
 ('mort', -0.51263319187574208),
 ('ren', -0.23342136375329145),
 ('fico', -0.086502602177937038)]
```

Now, we will have to penalise variables which are not contributing well to response and can cause overfitting as well. Therefore, lets apply Ridge Regression

Ridge

In [152]:

```
#Looking at multiple values of hyperparameter and using 10 fold cross validation
 choosing best one.

alphaa=np.linspace(.0001,10,100)
x_train.reset_index(drop=True,inplace=True)
y_train.reset_index(drop=True,inplace=True)
```

In [158]:

```
rmse_list=[]
for a in alphaa:
    ridge = Ridge(fit_intercept=True, alpha=a)
#Calculating average RMSE
    k = KFold(len(x_train), n_folds=10)
    xval_error = 0
    for train, test in k:
        ridge.fit(x_train.loc[train], y_train[train])
        b = ridge.predict(x_train.loc[test])
        error = b - y_train[test]
        xval_error += np.dot(error,error)
    rmse_cv = np.sqrt(xval_error/len(x_train))

    print('{:.3f}\t {:.6f}\t '.format(a,rmse_cv))
    rmse_list.extend([rmse_cv])
best_alpha=alphaa[rmse_list==min(rmse_list)]
print('Value of alpha having minimum 10 CV error is : ',best_alpha )
```

0.000	2.075747
0.101	2.075638
0.202	2.075556
0.303	2.075491
0.404	2.075439
0.505	2.075396
0.606	2.075360
0.707	2.075329
0.808	2.075303
0.909	2.075280
1.010	2.075259
1.111	2.075242
1.212	2.075226
1.313	2.075211
1.414	2.075198
1.515	2.075187
1.616	2.075176
1.717	2.075167
1.818	2.075158
1.919	2.075150
2.020	2.075143
2.121	2.075136
2.222	2.075130
2.323	2.075125
2.424	2.075120
2.525	2.075115
2.626	2.075111
2.727	2.075107
2.828	2.075104
2.929	2.075100
3.030	2.075098
3.131	2.075095
3.232	2.075093
3.333	2.075091
3.434	2.075090
3.535	2.075088
3.636	2.075087
3.737	2.075086
3.838	2.075086
3.939	2.075085
4.040	2.075085
4.141	2.075085
4.242	2.075085
4.343	2.075086
4.444	2.075086
4.546	2.075087
4.647	2.075088
4.748	2.075089
4.849	2.075091
4.950	2.075092
5.051	2.075094
5.152	2.075096
5.253	2.075098
5.354	2.075100
5.455	2.075102
5.556	2.075105
5.657	2.075107
5.758	2.075110
5.859	2.075113
5.960	2.075116
6.061	2.075119

```
6.162    2.075123
6.263    2.075126
6.364    2.075130
6.465    2.075134
6.566    2.075137
6.667    2.075141
6.768    2.075146
6.869    2.075150
6.970    2.075154
7.071    2.075159
7.172    2.075164
7.273    2.075168
7.374    2.075173
7.475    2.075178
7.576    2.075184
7.677    2.075189
7.778    2.075194
7.879    2.075200
7.980    2.075206
8.081    2.075211
8.182    2.075217
8.283    2.075223
8.384    2.075230
8.485    2.075236
8.586    2.075242
8.687    2.075249
8.788    2.075255
8.889    2.075262
8.990    2.075269
9.091    2.075276
9.192    2.075283
9.293    2.075290
9.394    2.075297
9.495    2.075305
9.596    2.075312
9.697    2.075320
9.798    2.075328
9.899    2.075335
10.000   2.075343
Value of alpha having minimum 10 CV error is :  [ 4.04046364]
```

best value of alpha might be slightly different across different runs because of random nature of cross validation. So dont worry if you determine a different value of best alpha.

Next we fit Ridge Regression on the entire train data with best value of alpha we just determined.

In [159]:

```
# With the best value of alpha, fitting ridge regression on our complete data se
t
ridge=Ridge(fit_intercept=True,alpha=best_alpha)
ridge.fit(x_train,y_train)
tst=ridge.predict(x_test)
residual=tst-y_test
rmse_ridge=np.sqrt(np.dot(residual,residual)/len(tst))
rmse_ridge
```

Out[159]:

```
1.9986610201010218
```

In [160]:

```
list(zip(x_train.columns,ridge.coef_))
```

Out[160]:

```
[('Amount.Requested', 0.00016586905207985439),
 ('Debt.To.Income.Ratio', 0.0020224200468194468),
 ('Monthly.Income', -2.0262579354427958e-05),
 ('Open.CREDIT.Lines', -0.03428997936472207),
 ('Revolving.CREDIT.Balance', -4.0012363985016265e-06),
 ('Inquiries.in.the.Last.6.Months', 0.35358237791769531),
 ('Employment.Length', 0.0060576014674007415),
 ('month_36', -3.0858882289105218),
 ('LP_cep', -0.060597535013371 27),
 ('LP_chos', 0.051904670459841672),
 ('LP_dg', -0.13915040742134219),
 ('LP_hmvg', -0.13894706764596293),
 ('mort', -0.48648146285694549),
 ('ren', -0.21080912056439927),
 ('fico', -0.08653038791149581)]
```

As can be seen, coefficients have been decreased but are not made 0. Now considering Lasso Regression.

Lasso

In [162]:

```
alphaa=np.linspace(0.0001,1,100)
rmse_list=[]
for a in alphaa:
    lasso = Lasso(fit_intercept=True, alpha=a,max_iter=10000)
# Finding RMSE
    k = KFold(len(x_train), n_folds=10)
    xval_error = 0
    for train, test in k:
        lasso.fit(x_train.loc[train], y_train[train])
        b =lasso.predict(x_train.loc[test])
        error = b - y_train[test]
        xval_error += np.dot(error,error)
    rmse_cv = np.sqrt(xval_error/len(x_train))
    rmse_list.extend([rmse_cv])

    print('{:.3f}\t {:.4f}\t '.format(a,rmse_cv))
best_alpha=alphaa[rmse_list==min(rmse_list)]
print('Value of alpha having minimum 10 CV error is: ',best_alpha )
```

```
0.000    2.0755
0.010    2.0747
0.020    2.0759
0.030    2.0774
0.041    2.0795
0.051    2.0823
0.061    2.0851
0.071    2.0878
0.081    2.0907
0.091    2.0939
0.101    2.0975
0.111    2.1015
0.121    2.1059
0.131    2.1107
0.141    2.1159
0.152    2.1214
0.162    2.1274
0.172    2.1337
0.182    2.1403
0.192    2.1474
0.202    2.1547
0.212    2.1625
0.222    2.1705
0.232    2.1789
0.242    2.1877
0.253    2.1968
0.263    2.2062
0.273    2.2160
0.283    2.2260
0.293    2.2364
0.303    2.2471
0.313    2.2582
0.323    2.2695
0.333    2.2811
0.343    2.2930
0.354    2.3052
0.364    2.3177
0.374    2.3305
0.384    2.3436
0.394    2.3569
0.404    2.3705
0.414    2.3844
0.424    2.3978
0.434    2.4056
0.444    2.4096
0.455    2.4111
0.465    2.4126
0.475    2.4141
0.485    2.4156
0.495    2.4169
0.505    2.4182
0.515    2.4193
0.525    2.4202
0.535    2.4207
0.545    2.4210
0.556    2.4211
0.566    2.4212
0.576    2.4212
0.586    2.4212
0.596    2.4212
0.606    2.4212
```

```
0.616    2.4211
0.626    2.4211
0.636    2.4211
0.646    2.4211
0.657    2.4211
0.667    2.4210
0.677    2.4210
0.687    2.4210
0.697    2.4210
0.707    2.4210
0.717    2.4210
0.727    2.4209
0.737    2.4209
0.747    2.4209
0.758    2.4209
0.768    2.4209
0.778    2.4209
0.788    2.4209
0.798    2.4209
0.808    2.4209
0.818    2.4209
0.828    2.4210
0.838    2.4210
0.848    2.4210
0.859    2.4210
0.869    2.4210
0.879    2.4210
0.889    2.4210
0.899    2.4210
0.909    2.4210
0.919    2.4210
0.929    2.4210
0.939    2.4210
0.949    2.4210
0.960    2.4210
0.970    2.4210
0.980    2.4210
0.990    2.4210
1.000    2.4210
Value of alpha having minimum 10 CV error is:  [ 0.0102]
```

In [163]:

```
lasso=Lasso(fit_intercept=True,alpha=best_alpha)
lasso.fit(x_train,y_train)
tst=lasso.predict(x_test)
residual=tst-y_test
rmse_lasso=np.sqrt(np.dot(residual,residual)/len(tst))
rmse_lasso
```

Out[163]:

```
1.9957102870584469
```

In [164]:

```
list(zip(x_train.columns,lasso.coef_))
```

Out[164]:

```
[('Amount.Requested', 0.00016596990419555173),
 ('Debt.To.Income.Ratio', 0.0018512121409904464),
 ('Monthly.Income', -2.1507229501854884e-05),
 ('Open.CREDIT.Lines', -0.033670424591482798),
 ('Revolving.CREDIT.Balance', -3.9690552293878563e-06),
 ('Inquiries.in.the.Last.6.Months', 0.34548495700631604),
 ('Employment.Length', 0.0043033318748095717),
 ('month_36', -3.0510561974874624),
 ('LP_cep', 0.0),
 ('LP_chos', 0.1155218594214963),
 ('LP_dg', -0.024266912121686985),
 ('LP_hmvg', -0.0),
 ('mort', -0.26583276808536699),
 ('ren', -0.0),
 ('fico', -0.086541751428909866)]
```

As can be seen Lasso Regression has improved the performance on data and also has made coefficients 0 which means it has made the model smaller.