PRACTICAL DATA SCIENCE

Project report

kiva.org

Team # 3
Suresh Rangarajan
Charlotte George
David Cohodes

12/19/2012

Summary

Data Source & Overview

URL: http://build.kiva.org/

Kiva provides a RESTful Web-service API for accessing data on lenders, loans and other related Kiva objects. Kiva API returns the response in XML and JSON formats depending on the URL accessed. We downloaded the bulk data provided by Kiva from the URL http://s3.kiva.org/snapshots/kiva_ds_json.zip to do an initial analysis of the data and came up with a few models and questions that we can answer with the data. In order to best apply the leanings from the Data Science class we used two methods to sort the data – Loading the data into MySQL database and pulling the data in as a JSON array using python. Using the SQL approach we installed MySQL community edition database on our laptop and created a training and test database with tables for the major kiva objects. We loaded the bulk data for lenders and loans using python. We also created python scripts to fetch some live data from Kiva API and loaded that data as well into the training and test databases to ensure that the models apply to old as well as new data. We also loaded other tables such as country, loan->lenders and lender -> loans to analyze the relationships between the objects. In the JSON In memory method, we loaded the JSON file attributes into python collection objects to locate specific attributes of loans made on kiva.org. We reused some of the data model from http://www.kivadata.org/ website for loading the data into MySQL database.

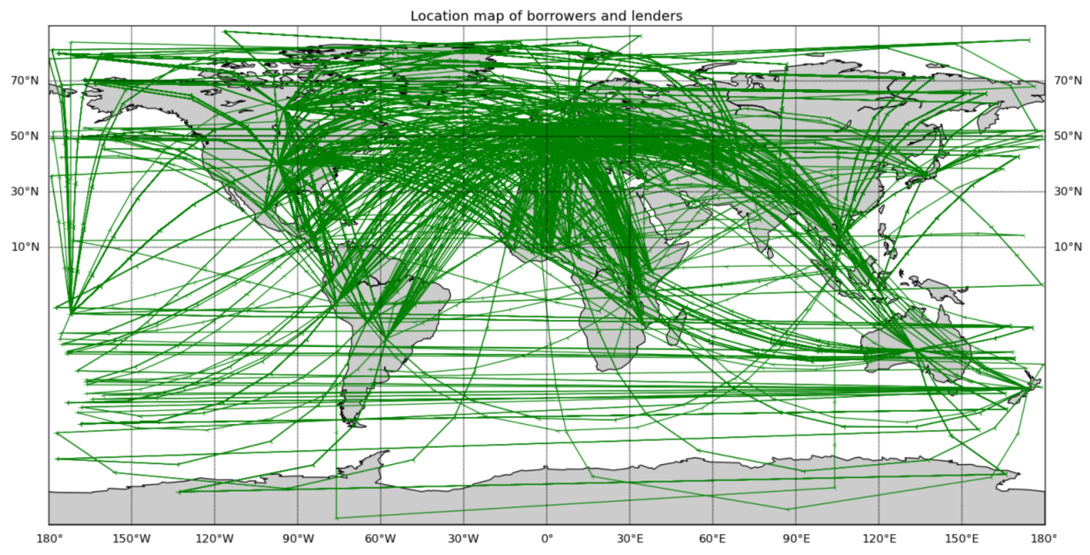
Analysis

1. Mapping locations of lenders and borrowers

We used the python map visualization library Basemap, to plot the locations of lenders and borrowers and then project the trend in kiva loans. Using the Kiva API we loaded all the lenders for a loan and used the country data from loan and lender objects to plot the map.

Python module: map_loan_countries.py

The most glaring aspect of this visualization is that Africa seems to be the region with the most borrowers and North America seems to be the region with most lenders, which is not surprising. South America and Mexico also have a high number of borrowers. Some of the South East Asian countries are also heavily active in kiva. Please note that the python library we used has a glitch which cannot draw the lines when the lines end and re-enter from the other side. These lines appear as straight lines in the bottom.



We could not do a mapping between lenders and borrower's demographics since data about lender demographics is not available except for their location and occupation.

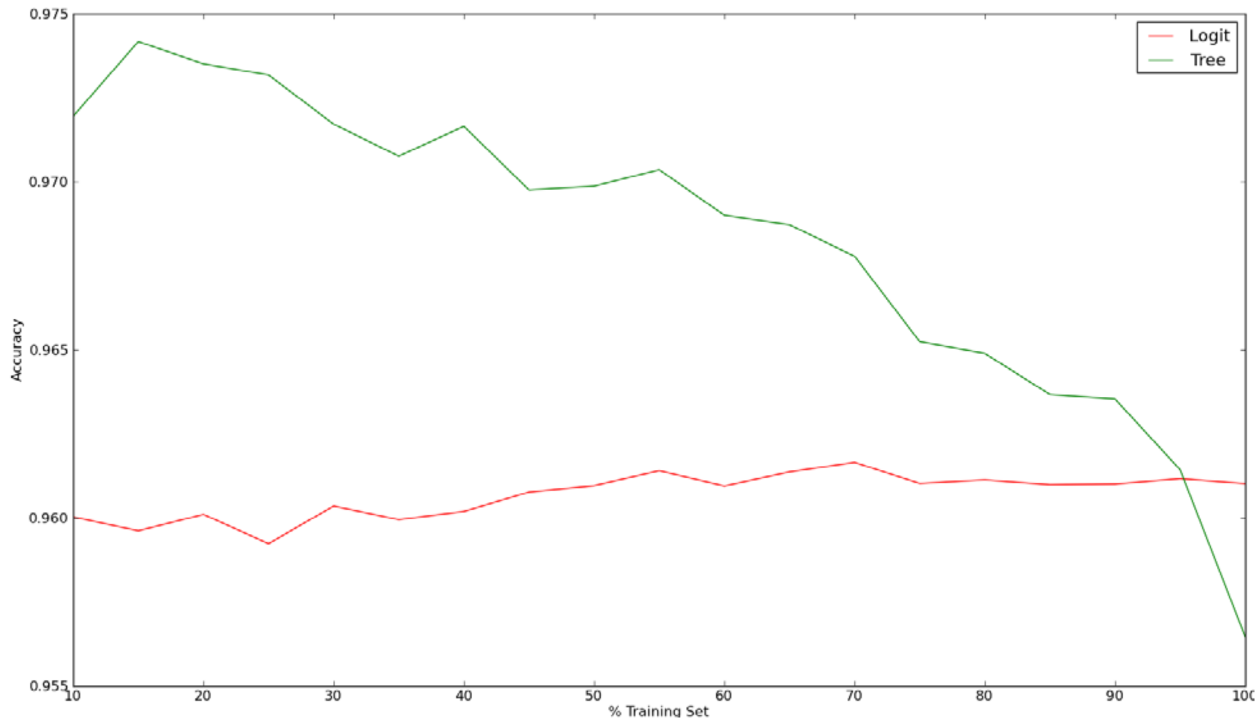
2. Predictive model to classify a specific loan as whether it will be fully funded or not

We used certain parameters such as loan_amount, paid_amount, funded_amount, disbursal_amount, sector, partner_id and gender as inputs to the predictive model to determine whether a particular loan will be fully funded or not. We realize that the factors used are small in number, related and do not capture all the parameters that would determine the possibility of a loan being funded.

However we used Logistic regression and decision tree models to determine the probability and used cross_validation to test the models with a varying size of training and test data sets.

Python module: model.py

The comparative plot of the accuracy of the models with the change in size of training and test data sets is shown below:



3. Analysis of the loans based on sector, activity, country and gender

SQL File : test.sql

We used the data in the training database and SQL queries to analyze the influence of sector, activity, country and gender in how loans are funded.

Following table lists the loan sectors ordered by % of fully funded loans:

Sector	Funded	Not funded	% Funded
Housing	9047	1175	88.50518
Personal Use	3074	248	92.53462
Clothing	22617	1276	94.65952
Transportation	11064	599	94.8641
Retail	75379	4026	94.92979
Agriculture	67459	3419	95.17622
Services	25878	1193	95.59307
Food	85674	3610	95.95672
Wholesale	805	32	96.17682
Construction	6478	211	96.84557
Health	2734	79	97.19161
Entertainment	599	15	97.557
Education	2838	70	97.59285
Arts	7389	167	97.78984
Manufacturing	4601	100	97.87279

Housing, personal use and clothing sectors attract lower fraction of loans while manufacturing, arts, education and healthcare attract relatively higher fraction of loans.

For an individual looking for a loan these are important statistics to understand. If you take as a starting point that the borrower is looking to better his or her standard of living by whatever means is most efficient, then it makes more sense for borrowers to pursue businesses where more loans are made (assuming the market is not oversaturated in these areas). If the market is saturated in these more common areas, then Kiva is still useful as it allows long tail borrowers (those with less common lending needs) to still receive a funded loan.

By country

Appendix 3 lists the loans to borrowers in each country. Countries such as Gaza, Timor-Leste and Turkey have a very high proportion of funded loans. In contrast, Bangladesh, Jordan and Zambia receive a low fraction of fully funded loans.

Following table lists the number of loans and proportion of funded loans by country for the top 10 countries with most number of loans:

Country	Funded	Not funded	%Funded	Total loans
GH	9076	185	98.0024	9261
MX	8223	189	97.7532	8412
PE	35826	1085	97.0605	36911
PH	49078	1573	96.8944	50651
KE	26070	1001	96.3023	27071
KH	22730	1032	95.6569	23762
NI	15924	923	94.5213	16847
EC	9379	602	93.9685	9981
UG	11099	837	92.9876	11936
TJ	10468	915	91.9617	11383

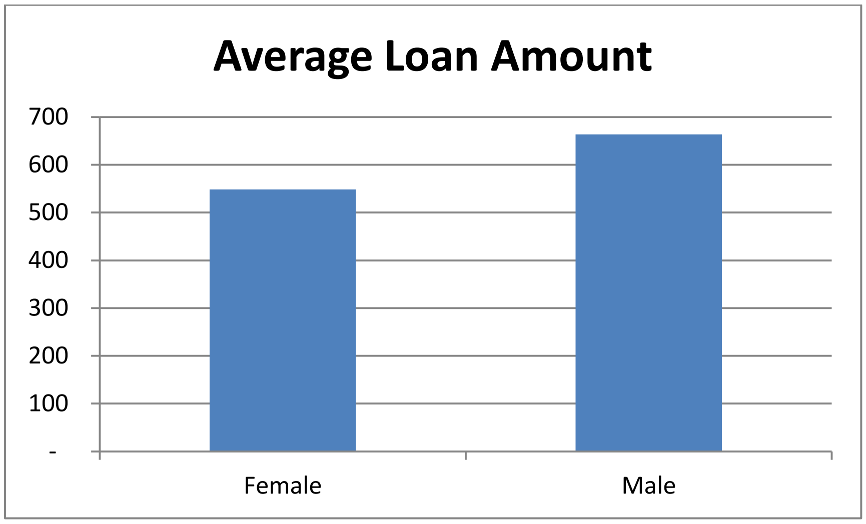
Among the top 10 countries with most number of loans, Ghana and Mexico have a high ratio of funded loans. Tajikistan and Uganda have a low ratio in the top 10 countries.

By gender

The following table lists the number of loans and the proportion of funded loans by gender.

Gender	Not funded	Funded	%Funded
F	7958	205389	96.26993
M	5329	77520	93.56782
N	2933	42727	93.57643

Female borrowers attract approximately 3% more loans than male borrowers or borrowers of unknown gender. This shows a clear inclination towards lending to woman. However on average the amount of money that men are loaned is much higher.



By activity

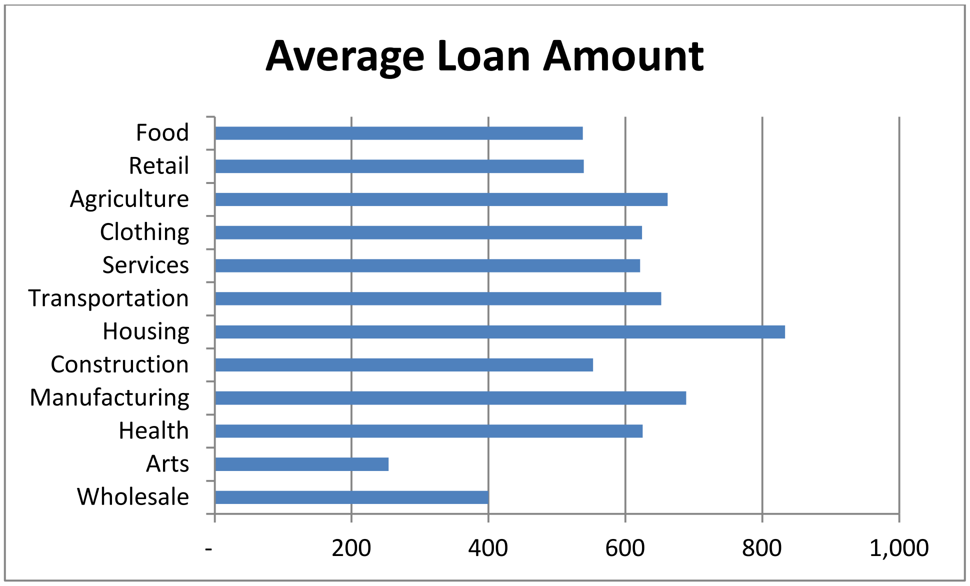
There are 149 activities listed under the 15 sectors in Kiva. Appendix 4 lists the number of loans and proportion of funded loans by activity. Activities such as Machine Shop, Well digging and Renewable Energy Products attract very high proportion of funding. Activities such as Personal Housing Expenses, Funeral Expenses, Machinery Rental and Wedding Expenses attract very low proportion of funding.

Following table lists the top 10 activities with most number of loans along with the proportions:

Activity	**Not funded**	**Funded**	**% Funded**	**Total**
Food Production/Sales	627	17426	96.52689	18053
Fruits & Vegetables	350	8983	96.24987	9333
Food Market	416	9814	95.93353	10230
General Store	1103	25464	95.84823	26567
Agriculture	609	13372	95.64409	13981
Grocery Store	605	13087	95.58136	13692
Farming	1249	22978	94.84459	24227
Clothing Sales	1036	17477	94.40393	18513
Retail	1417	22753	94.13736	24170
Personal Housing Expenses	1132	8207	87.87879	9339

Within the top 10 funded activates Food Production/Sales and Fruits & Vegetables activities attract a high proportion of funding while Retail and Personal Housing Expenses activities attract a lower proportion.

4. Loan Amount

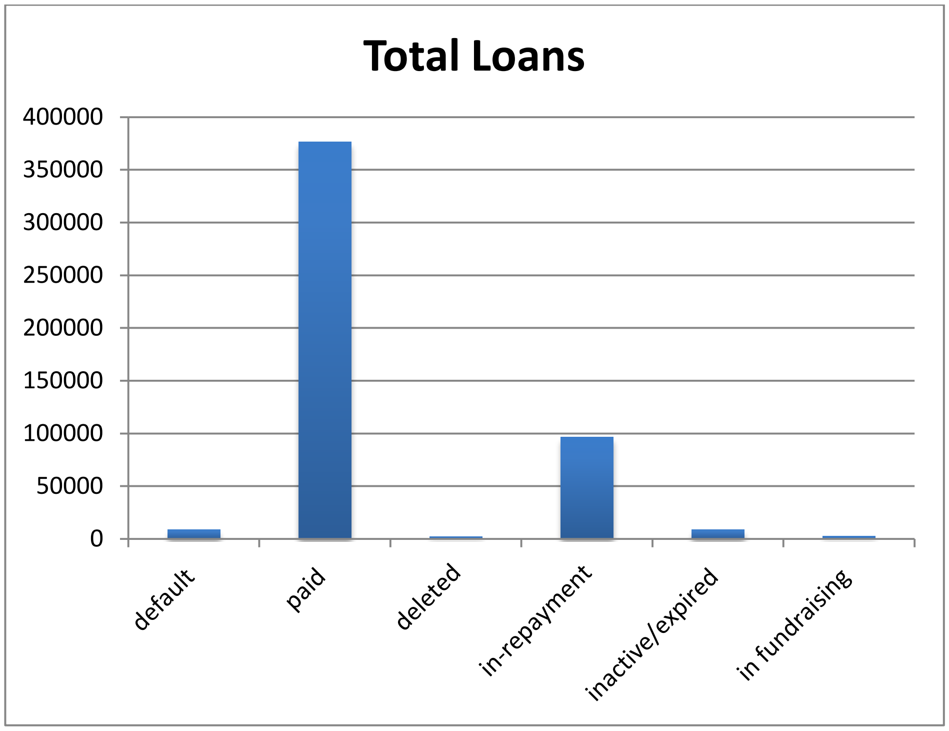


What this data shows is that the average dollar amount of a loan is not necessarily dependent on the popularity of the sector. For instance, the average loan in the agricultural sector is almost the same as the average loan in the manufacturing sector even though agricultural loans are much more popular.

If we compute the overall average dollar amount per loan, it is $577, which is much higher than the micro loans that are typically advertised on Kiva, but much lower than a loan that a bank would typically make. Clearly, this data shows that these borrowers are receiving loans that they normally would not have access to.

5. Default of Loans

In the data we looked at there were a total of 507,403 entries of loans. Kiva classifies the loans by the categories shows in the below graph. Approximately 1.7% of the loans in this data set have defaulted.



In order to better understand where the defaults were coming from we identified the countries with the highest default percentage and the most number of defaults, as show below. This data would be helpful to lenders when considering who/where to lend to.

Countries with > 1% of loans defaulted

Country	% of defaults	total loans made
Afganistan	23.44	2474
Liberia	17.73	3751
Dominican Republic	13.84	3757
Tanzania	9.26	9447
Togo	6.32	9480
Kenya	5.49	40114
Ecuador	4.49	14698
Zimbabwe	4.145	534

Sierra Leone	3.976	6137
Guatemala	2.62	3433
Turkey	2.72	44
Bulgaria	2.02	296
Burkina Faso	1.81	386
Haiti	1.786	224
Zambia	1.67	60
Mexico	1.178	12216
Uganda	1.09	17913

Countries with above 100 defaulted loans

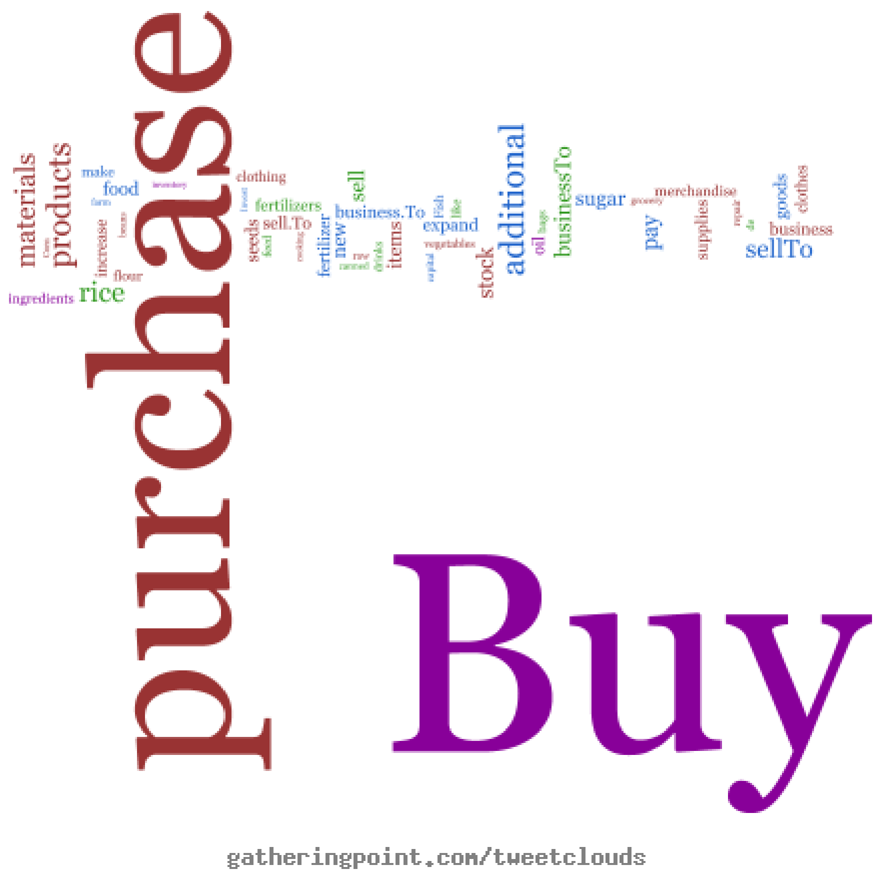
Country	# of defaults
Kenya	2537
Tanzania	875
Togo	873
Ecuador	807
Liberia	665
Afghanistan	580
Dominican Republic	520
Peru	307
Sierra Leone	244
Uganda	196
Nicaragua	187
Mexico	144
Ghana	107

<u>6. Word cloud of lender loaning reason and loan uses:</u>

In order to find more about what were the most important reasons for a lender providing a loan and the most frequent uses of loans we used the text provided in the loans and lender profiles to create word clouds.

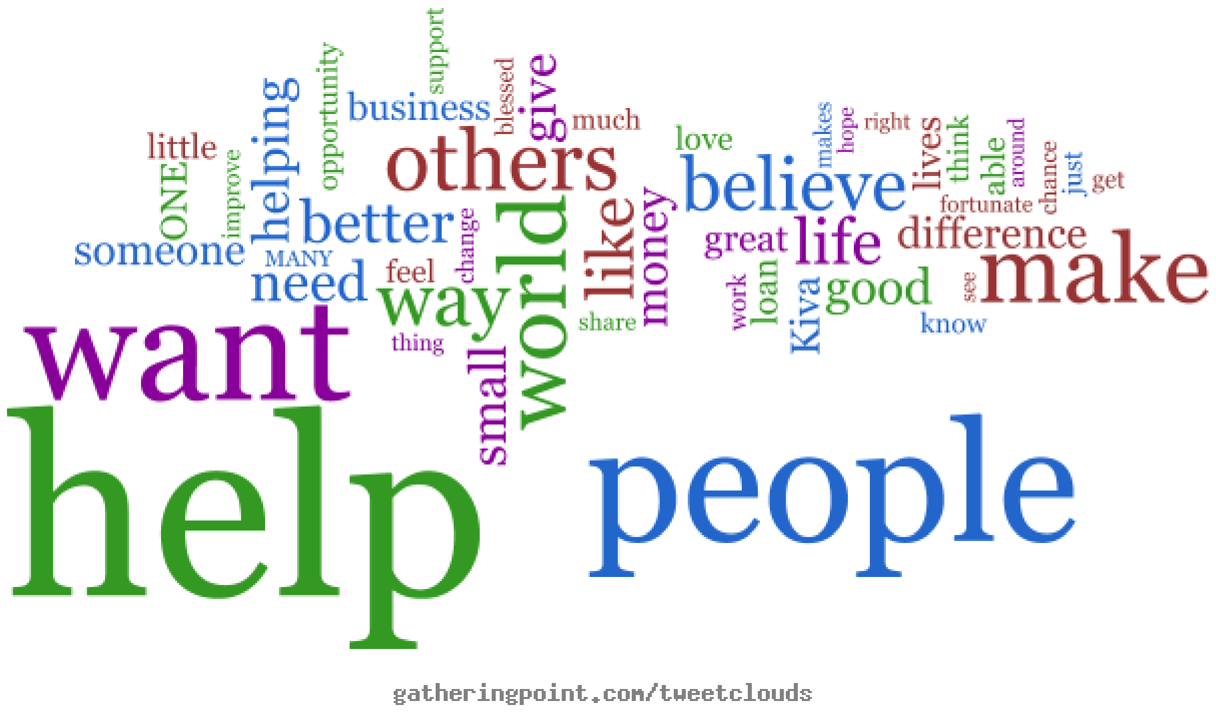
Python module: WordCloudsample.py

Following is the word cloud of the uses of loans:



It is clear that borrowers like to purchase or buy items like rice, sugar, clothing materials.

Following is the word cloud of lender's reasons for providing a loan.



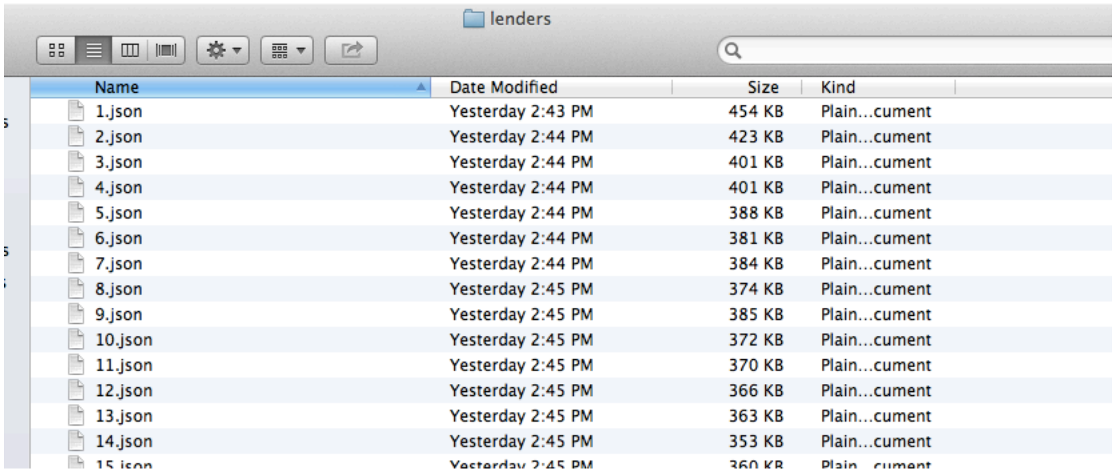
It is clear that lenders want to help people and want to make a difference and believe in others.

Project Breakdown:

Suresh wrote the code to pull data into MySQL and used this to create predictive models and generate maps. Charlotte wrote the code to pull data in using Python to complete the loan default aspect. David utilized the python code to create visualizations and analyze data in regards to activity, sector and characteristics.

Data Format:

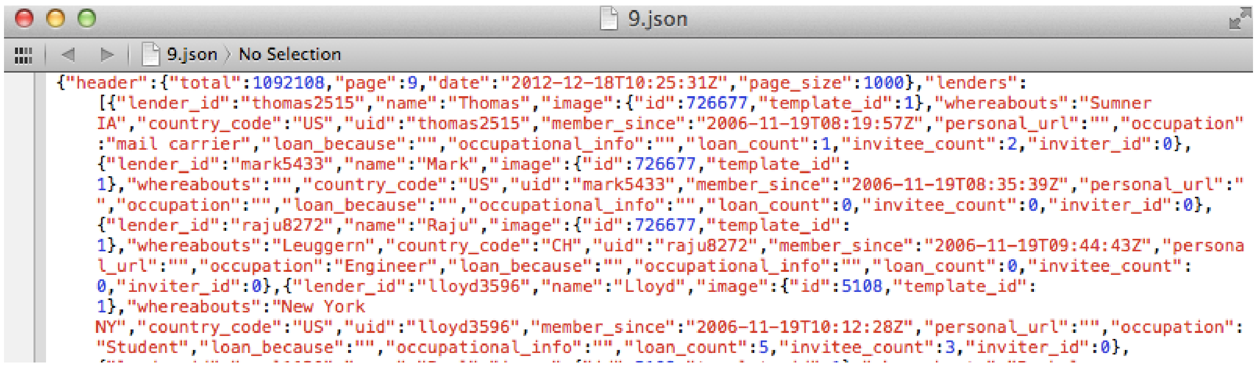
We used the data in JSON format, which was broken down into smaller files of "lenders" and "loans".



lenders

Name	Date Modified	Size	Kind
1.json	Yesterday 2:43 PM	454 KB	Plain...cument
2.json	Yesterday 2:44 PM	423 KB	Plain...cument
3.json	Yesterday 2:44 PM	401 KB	Plain...cument
4.json	Yesterday 2:44 PM	401 KB	Plain...cument
5.json	Yesterday 2:44 PM	388 KB	Plain...cument
6.json	Yesterday 2:44 PM	381 KB	Plain...cument
7.json	Yesterday 2:44 PM	384 KB	Plain...cument
8.json	Yesterday 2:45 PM	374 KB	Plain...cument
9.json	Yesterday 2:45 PM	385 KB	Plain...cument
10.json	Yesterday 2:45 PM	372 KB	Plain...cument
11.json	Yesterday 2:45 PM	370 KB	Plain...cument
12.json	Yesterday 2:45 PM	366 KB	Plain...cument
13.json	Yesterday 2:45 PM	363 KB	Plain...cument
14.json	Yesterday 2:45 PM	353 KB	Plain...cument
15.json	Yesterday 2:45 PM	360 KB	Plain...cument

Each JSON file contained a JSON array with 300-500 entries within



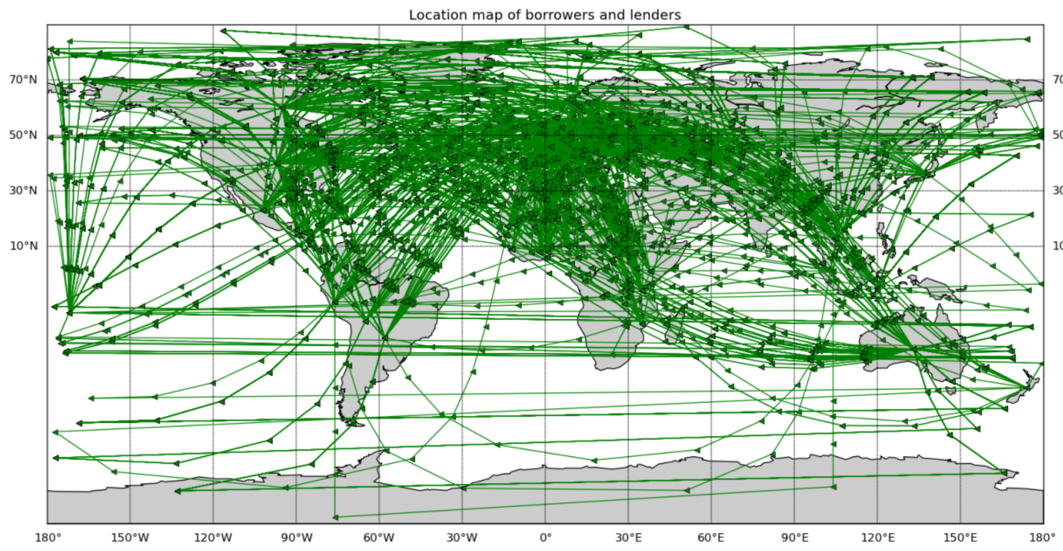
9.json

9.json 〉 No Selection

{"header":{"total":1092108,"page":9,"date":"2012-12-18T10:25:31Z","page_size":1000},"lenders":
    [{"lender_id":"thomas2515","name":"Thomas","image":{"id":726677,"template_id":1},"whereabouts":"Sumner
    IA","country_code":"US","uid":"thomas2515","member_since":"2006-11-19T08:19:57Z","personal_url":"","occupation"
    :"mail carrier","loan_because":"","occupational_info":"","loan_count":1,"invitee_count":2,"inviter_id":0},
    {"lender_id":"mark5433","name":"Mark","image":{"id":726677,"template_id":
    1},"whereabouts":"","country_code":"US","uid":"mark5433","member_since":"2006-11-19T08:35:39Z","personal_url":"
    ","occupation":"","loan_because":"","occupational_info":"","loan_count":0,"invitee_count":0,"inviter_id":0},
    {"lender_id":"raju8272","name":"Raju","image":{"id":726677,"template_id":
    1},"whereabouts":"Leuggern","country_code":"CH","uid":"raju8272","member_since":"2006-11-19T09:44:43Z","persona
    l_url":"","occupation":"Engineer","loan_because":"","occupational_info":"","loan_count":0,"invitee_count":
    0,"inviter_id":0},{"lender_id":"lloyd3596","name":"Lloyd","image":{"id":5108,"template_id":
    1},"whereabouts":"New York
    NY","country_code":"US","uid":"lloyd3596","member_since":"2006-11-19T10:12:28Z","personal_url":"","occupation":
    "Student","loan_because":"","occupational_info":"","loan_count":5,"invitee_count":3,"inviter_id":0},

<u>Appendix 1:</u>

Alternative visualization of location of lenders and borrowers.



<u>Appendix 2:</u>

Output from model.py module, which used logistic regression and decision tree models to predict the possibility of a loan being fully funded:

```
Number of train records: 171101
Number of test records: 74205
length: 171101
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records: 
171101
Accuracy :0.960888597963
length: 171101
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 171101
Accuracy :0.99722386193
----------------------------------------------------------------------
Evaluating for : 5 %
----------------------------------------------------------------------
Training data#:  162545  Test data#:  8556
length: 8556
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records: 
8556
Accuracy :0.960028050491
length: 8556
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 8556
Accuracy :0.971949509116
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 10 %
----------------------------------------------------------------------
```

```
Training data#:  153990  Test data#:  17111
length: 17111
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
17111
Accuracy :0.959616620887
length: 17111
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 17111
Accuracy :0.974168663433
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 15 %
----------------------------------------------------------------------
Training data#:  145435  Test data#:  25666
length: 25666
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
25666
Accuracy :0.960102859815
length: 25666
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 25666
Accuracy :0.973505805346
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 20 %
----------------------------------------------------------------------
Training data#:  136880  Test data#:  34221
length: 34221
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
34221
Accuracy :0.959235557114
```

```
length: 34221
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 34221
Accuracy :0.973174366617
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 25 %
---------------------------------------------------------------------
Training data#:  128325  Test data#:  42776
length: 42776
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
42776
Accuracy :0.960351599027
length: 42776
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 42776
Accuracy :0.971713110155
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 30 %
---------------------------------------------------------------------
Training data#:  119770  Test data#:  51331
length: 51331
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
51331
Accuracy :0.959946231322
length: 51331
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 51331
Accuracy :0.970758411097
---------------------------------------------------------------------
---------------------------------------------------------------------
```

```
Evaluating for : 35 %
------------------------------------------------------------------------
Training data#:  111215  Test data#:  59886
length: 59886
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
59886
Accuracy :0.960191029623
length: 59886
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 59886
Accuracy :0.971646127643
------------------------------------------------------------------------
------------------------------------------------------------------------
Evaluating for : 40 %
------------------------------------------------------------------------
Training data#:  102660  Test data#:  68441
length: 68441
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
68441
Accuracy :0.960769129615
length: 68441
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 68441
Accuracy :0.969754971435
------------------------------------------------------------------------
------------------------------------------------------------------------
Evaluating for : 45 %
------------------------------------------------------------------------
Training data#:  94105  Test data#:  76996
length: 76996
```

```
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
76996
Accuracy :0.960959010858
length: 76996
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 76996
Accuracy :0.969868564601
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 50 %
---------------------------------------------------------------------
Training data#:  85550  Test data#:  85551
length: 85551
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
85551
Accuracy :0.961403139648
length: 85551
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 85551
Accuracy :0.970356863158
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 55 %
---------------------------------------------------------------------
Training data#:  76995  Test data#:  94106
length: 94106
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
94106
Accuracy :0.960948292351
length: 94106
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 94106
```

```
Accuracy :0.969003039126
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 60 %
----------------------------------------------------------------------
Training data#:  68440  Test data#:  102661
length: 102661
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
102661
Accuracy :0.961367997584
length: 102661
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 102661
Accuracy :0.968722299607
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 65 %
----------------------------------------------------------------------
Training data#:  59885  Test data#:  111216
length: 111216
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
111216
Accuracy :0.961651201266
length: 111216
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 111216
Accuracy :0.967783412459
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 70 %
----------------------------------------------------------------------
```

```
Training data#:  51330  Test data#:  119771
length: 119771
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
119771
Accuracy :0.961025623899
length: 119771
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 119771
Accuracy :0.96524200349
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 75 %
---------------------------------------------------------------------
Training data#:  42775  Test data#:  128326
length: 128326
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
128326
Accuracy :0.961130246404
length: 128326
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 128326
Accuracy :0.964894097845
---------------------------------------------------------------------
---------------------------------------------------------------------
Evaluating for : 80 %
---------------------------------------------------------------------
Training data#:  34220  Test data#:  136881
length: 136881
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
136881
Accuracy :0.960988011484
```

```
length: 136881
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 136881
Accuracy :0.963669172493
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 85 %
----------------------------------------------------------------------
Training data#:  25665  Test data#:  145436
length: 145436
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
145436
Accuracy :0.96100690338
length: 145436
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 145436
Accuracy :0.963537225996
----------------------------------------------------------------------
----------------------------------------------------------------------
Evaluating for : 90 %
----------------------------------------------------------------------
Training data#:  17110  Test data#:  153991
length: 153991
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
153991
Accuracy :0.961166561682
length: 153991
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 153991
Accuracy :0.961432811008
----------------------------------------------------------------------
----------------------------------------------------------------------
```

```
Evaluating for : 95 %
----------------------------------------------------------------------
Training data#:  8555  Test data#:  162546
length: 162546
Model:  <class 'sklearn.linear_model.logistic.LogisticRegression'> No of test records:
162546
Accuracy :0.961020265033
length: 162546
Model:  <class 'sklearn.tree.tree.DecisionTreeClassifier'> No of test records: 162546
Accuracy :0.956461555498
```

Appendix 3:

Table listing loans to borrowers in each country

Country	Funded	Not funded	%Funded	Total loans
GZ	8	0	100	8
TL	77	0	100	77
TR	8	0	100	8
BG	209	1	99.5238	210
CI	170	1	99.4152	171
BA	316	3	99.0596	319
UA	2297	25	98.9233	2322
CD	706	13	98.1919	719
DO	2605	51	98.0798	2656
GH	9076	185	98.0024	9261
QS	3724	80	97.897	3804
MX	8223	189	97.7532	8412
PY	5650	134	97.6833	5784
CL	495	12	97.6331	507
IL	112	3	97.3913	115
CG	435	12	97.3154	447
HT	143	4	97.2789	147
RW	4507	129	97.2174	4636
MZ	1708	49	97.2112	1757
PE	35826	1085	97.0605	36911
KG	2306	72	96.9722	2378
ZA	128	4	96.9697	132
TG	6313	199	96.9441	6512
VN	3676	117	96.9154	3793

PH	49078	1573	96.8944	50651
LR	2450	81	96.7997	2531
PK	4811	161	96.7619	4972
ID	2232	75	96.749	2307
NP	831	28	96.7404	859
KE	26070	1001	96.3023	27071
ZW	349	15	95.8791	364
GT	2191	95	95.8443	2286
KH	22730	1032	95.6569	23762
MD	227	11	95.3782	238
NG	5842	296	95.1776	6138
CR	1425	74	95.0634	1499
US	460	24	95.0413	484
SL	3883	207	94.9389	4090
MN	4381	236	94.8885	4617
BJ	3225	175	94.8529	3400
SN	4502	248	94.7789	4750
BI	285	16	94.6844	301
NI	15924	923	94.5213	16847
WS	3974	238	94.3495	4212
BO	7811	469	94.3357	8280
LB	4816	290	94.3204	5106
AF	1613	98	94.2724	1711
EC	9379	602	93.9685	9981
ML	2921	196	93.7119	3117
CM	895	62	93.5214	957
PS	1692	119	93.429	1811
IN	141	10	93.3775	151

AZ	4649	333	93.3159	4982
UG	11099	837	92.9876	11936
LK	188	15	92.6108	203
HN	3442	293	92.1553	3735
TJ	10468	915	91.9617	11383
SV	7341	642	91.9579	7983
YE	399	36	91.7241	435
TZ	6065	580	91.2716	6645
GE	1055	101	91.263	1156
BF	207	20	91.1894	227
TD	40	4	90.9091	44
XK	138	15	90.1961	153
AL	164	22	88.172	186
CO	3420	462	88.0989	3882
AM	1251	173	87.8511	1424
IQ	1008	151	86.9715	1159
ZM	29	8	78.3784	37
JO	1817	878	67.4212	2695
BD	0	12	0	12

Appendix 4:

List of loans and proportion of funded loans by activity

Activity	Not funded	Funded	% Funded	Total
Bookbinding	0	13	100	13
Film	0	8	100	8
Machine Shop	0	64	100	64
Renewable Energy Products	0	27	100	27
Well digging	0	29	100	29
Land Rental	1	79	98.75	80
Secretarial Services	2	156	98.73418	158
Weaving	28	1949	98.58371	1977
Bicycle Sales	1	68	98.55072	69
Internet Cafe	8	537	98.53211	545
Musical Performance	2	126	98.4375	128
Souvenir Sales	3	174	98.30508	177
Call Center	4	231	98.29787	235
Metal Shop	12	678	98.26087	690
Child Care	2	108	98.18182	110
Tourism	2	103	98.09524	105
Timber Sales	11	547	98.02867	558
Primary/secondary school costs	25	1224	97.9984	1249
Bicycle Repair	5	243	97.98387	248
Entertainment	5	238	97.94239	243
Blacksmith	10	456	97.85408	466
Manufacturing	40	1796	97.82135	1836
Natural Medicines	13	582	97.81513	595

Recycled Materials	5	216	97.73756	221
Embroidery	21	904	97.72973	925
Furniture Making	38	1607	97.68997	1645
Knitting	10	412	97.63033	422
Construction	54	2223	97.62846	2277
Pharmacy	31	1272	97.62087	1303
Crafts	70	2864	97.61418	2934
Traveling Sales	12	483	97.57576	495
Musical Instruments	2	79	97.53086	81
Bookstore	12	472	97.52066	484
Dairy	93	3609	97.48784	3702
Arts	13	497	97.45098	510
Phone Repair	3	113	97.41379	116
Water Distribution	8	301	97.411	309
Bricks	13	484	97.38431	497
Hotel	6	223	97.37991	229
Education provider	12	433	97.30337	445
Dental	3	108	97.2973	111
Higher education costs	33	1181	97.28171	1214
Carpentry	31	1109	97.2807	1140
Cheese Making	7	242	97.18876	249
Laundry	7	242	97.18876	249
Printing	10	341	97.151	351
Recycling	14	471	97.1134	485
Medical Clinic	8	266	97.08029	274
Computers	13	432	97.07865	445
Bakery	95	3014	96.94436	3109
Patchwork	2	63	96.92308	65

Fishing	82	2497	96.82047	2579
Fish Selling	205	6224	96.81132	6429
Cloth & Dressmaking Supplies	53	1579	96.75245	1632
Cobbler	17	504	96.73704	521
Goods Distribution	18	533	96.73321	551
Motorcycle Repair	12	353	96.71233	365
Games	8	235	96.70782	243
Textiles	21	613	96.6877	634
Photography	14	407	96.67458	421
Catering	49	1413	96.64843	1462
Sewing	132	3794	96.6378	3926
Fuel/Firewood	57	1625	96.61118	1682
Food Production/Sales	627	17426	96.52689	18053
Pigs	257	7024	96.47027	7281
Tailoring	153	4141	96.43689	4294
Balut-Making	1	27	96.42857	28
Fruits & Vegetables	350	8983	96.24987	9333
Motorcycle Transport	185	4744	96.2467	4929
Cereals	110	2809	96.23159	2919
Charcoal Sales	131	3275	96.15385	3406
Religious Articles	4	99	96.1165	103
Milk Sales	31	766	96.11041	797
Jewelry	38	921	96.03754	959
Used Clothing	149	3557	95.97949	3706
Waste Management	4	95	95.9596	99
Restaurant	165	3900	95.94096	4065
Poultry	144	3403	95.94023	3547

Food Market	416	9814	95.93353	10230
General Store	1103	25464	95.84823	26567
Phone Use Sales	15	345	95.83333	360
Butcher Shop	87	1953	95.73529	2040
Agriculture	609	13372	95.64409	13981
Grocery Store	605	13087	95.58136	13692
Used Shoes	10	215	95.55556	225
Office Supplies	16	342	95.53073	358
Personal Medical Expenses	11	232	95.47325	243
Quarrying	11	232	95.47325	243
Transportation	132	2783	95.4717	2915
Health	13	274	95.47038	287
Food Stall	218	4587	95.46306	4805
Construction Supplies	83	1714	95.38119	1797
Electrician	12	247	95.3668	259
Liquor Store / Off-License	45	924	95.35604	969
Rickshaw	25	513	95.35316	538
Perfumes	14	284	95.30201	298
Beauty Salon	205	4142	95.2841	4347
Barber Shop	40	808	95.28302	848
Flowers	17	333	95.14286	350
Property	43	840	95.13024	883
Wholesale	14	272	95.1049	286
Shoe Sales	139	2663	95.03926	2802
Hardware	39	734	94.95472	773
Soft Drinks	115	2129	94.87522	2244
Electronics Repair	22	407	94.87179	429
Sporting Good Sales	2	37	94.87179	39

Farming	1249	22978	94.84459	24227
Utilities	9	165	94.82759	174
Animal Sales	322	5842	94.77612	6164
Veterinary Sales	6	106	94.64286	112
Plastics Sales	30	529	94.63327	559
Cement	8	140	94.59459	148
Pub	48	834	94.55782	882
Clothing	81	1368	94.40994	1449
Clothing Sales	1036	17477	94.40393	18513
Decorations Sales	18	301	94.35737	319
Vehicle	82	1370	94.35262	1452
Livestock	269	4399	94.23736	4668
Retail	1417	22753	94.13736	24170
Movie Tapes & DVDs	19	296	93.96825	315
Cosmetics Sales	261	4052	93.94853	4313
Music Discs & Tapes	10	154	93.90244	164
Spare Parts	83	1276	93.89257	1359
Services	329	5054	93.88817	5383
Personal Products Sales	92	1383	93.76271	1475
Auto Repair	82	1226	93.73089	1308
Electronics Sales	38	567	93.71901	605
Cattle	301	4446	93.65915	4747
Vehicle Repairs	57	823	93.52273	880
Food	251	3624	93.52258	3875
Upholstery	7	101	93.51852	108
Home Energy	13	180	93.26425	193
Cafe	103	1421	93.24147	1524
Home Products Sales	253	3461	93.18794	3714

Paper Sales	30	380	92.68293	410
Farm Supplies	151	1868	92.52105	2019
Personal Purchases	81	999	92.5	1080
Mobile Phones	45	534	92.22798	579
Electrical Goods	31	365	92.17172	396
Taxi	257	3024	92.16702	3281
Phone Accessories	38	386	91.03774	424
Consumer Goods	16	147	90.18405	163
Air Conditioning	5	41	89.13043	46
Party Supplies	17	134	88.74172	151
Home Appliances	35	268	88.44884	303
Personal Housing Expenses	1132	8207	87.87879	9339
Funeral Expenses	1	7	87.5	8
Machinery Rental	9	56	86.15385	65
Wedding Expenses	20	103	83.73984	123

Appendix 5:

createdb.sql

```
CREATE TABLE `lender` (
  `lender_id` varchar(100) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  `image_id` int(11) DEFAULT NULL,
  `template_id` int(11) DEFAULT NULL,
  `whereabouts` varchar(100) DEFAULT NULL,
  `country_code` varchar(100) DEFAULT NULL,
  `lender_uid` varchar(100) DEFAULT NULL,
  `member_since` date DEFAULT NULL,
  `personal_url` varchar(4000) DEFAULT NULL,
  `occupation` varchar(4000) DEFAULT NULL,
  `loan_because` text,
  `occupational_info` text,
  `loan_count` int(11) DEFAULT NULL,
  `invitee_count` int(11) DEFAULT NULL,
  PRIMARY KEY (`lender_id`)
) ;
CREATE TABLE `loan` (
  `id` int(11) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
  `status` varchar(100) DEFAULT NULL,
  `funded_amount` int(11) DEFAULT NULL,
  `paid_amount` int(11) DEFAULT NULL,
  `image_id` int(11) DEFAULT NULL,
  `template_id` int(11) DEFAULT NULL,
  `activity` varchar(100) DEFAULT NULL,
  `sector` varchar(100) DEFAULT NULL,
  `uses` varchar(4000) DEFAULT NULL,
  `country` varchar(100) DEFAULT NULL,
  `town` varchar(100) DEFAULT NULL,
  `geoLevel` varchar(100) DEFAULT NULL,
  `geoPairs` varchar(100) DEFAULT NULL,
  `geoType` varchar(100) DEFAULT NULL,
  `partner_id` int(11) DEFAULT NULL,
  `disbursal_amount` int(11) DEFAULT NULL,
  `disbursal_currency` varchar(100) DEFAULT NULL,
  `disbursal_date` date DEFAULT NULL,
  `loan_amount` int(11) DEFAULT NULL,
  `nonpayment` varchar(100) DEFAULT NULL,
  `currency_exchange` varchar(100) DEFAULT NULL,
  `posted_date` date DEFAULT NULL,
  `funded_date` date DEFAULT NULL,
  `defaulted_date` date DEFAULT NULL,
  `paid_date` date DEFAULT NULL,
  `refunded_date` date DEFAULT NULL,
  `journal_entries` int(11) DEFAULT NULL,
  `journal_bulk_entries` int(11) DEFAULT NULL,
  `gender` varchar(1) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ;
CREATE TABLE `partner` (
  `id` int(11) NOT NULL,
  `name` varchar(4000) DEFAULT NULL,
  `status` varchar(100) DEFAULT NULL,
  `rating` int(11) DEFAULT NULL,
  `image_id` int(11) DEFAULT NULL,
  `template_id` int(11) DEFAULT NULL,
  `start_date` date DEFAULT NULL,
  `delinquency_rate` int(11) DEFAULT NULL,
  `default_rate` int(11) DEFAULT NULL,
  `total_amount_raised` int(11) DEFAULT NULL,
  `loans_posted` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ;
CREATE TABLE `country` (
  `name` varchar(100) NOT NULL,
  `iso_code` varchar(2) DEFAULT NULL,
  `region` varchar(100) DEFAULT NULL,
  `latitude` int(11) DEFAULT NULL,
  `longitude` int(11) DEFAULT NULL,
  `gdpPerCapitaPPP` int(11) DEFAULT NULL,
  PRIMARY KEY (`name`),
  KEY `fk_country_region` (`region`)
) ;

CREATE TABLE `lender_loans` (
  `lender_id` varchar(100) DEFAULT NULL,
  `loan_id` int(11) DEFAULT NULL,
  KEY `loan_id` (`loan_id`),
  KEY `lender_id` (`lender_id`),
  CONSTRAINT `lender_loans_ibfk_1` FOREIGN KEY (`loan_id`) REFERENCES `loan` (`id`),
  CONSTRAINT `lender_loans_ibfk_2` FOREIGN KEY (`lender_id`) REFERENCES `lender` (`lender_id`)
) ;
CREATE TABLE `loan_lenders` (
  `loan_id` int(11) DEFAULT NULL,
  `lender_id` varchar(100) DEFAULT NULL,
  KEY `loan_id` (`loan_id`),
  KEY `lender_id` (`lender_id`),
  CONSTRAINT `loan_lenders_ibfk_1` FOREIGN KEY (`loan_id`) REFERENCES `loan` (`id`),
  CONSTRAINT `loan_lenders_ibfk_2` FOREIGN KEY (`lender_id`) REFERENCES `lender` (`lender_id`)
) ;
```

map_loan_countries.py

```
import MySQLdb
import httplib2
import json
import _mysql_exceptions
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt



db = MySQLdb.connect("localhost","root","password","kiva" )

cursor = db.cursor()
cursor1 = db.cursor()
cursor.execute("select * from loan_lenders where rand()<= 0.02")
data = cursor.fetchall()
fig=plt.figure()
ax=fig.add_axes([0.1,0.1,0.8,0.8])
m = Basemap(projection='cyl')
for row in data:
    cursor1.execute("select latitude,longitude from country where iso_code=(select country from loan where id
='"+str(row[0])+"')")
    loancountry = cursor1.fetchall()
    cursor1.execute("select latitude,longitude from country where iso_code=(select country_code from lender
where lender_id ='"+str(row[1])+"')")
    lendercountry = cursor1.fetchall()
    loanlat=0 ;loanlon=0; lenderlat=0;lenderlon=0
    for row in loancountry:
        loanlat = row[0]; loanlon = row[1]
    for row in lendercountry:
        lenderlat = row[0]; lenderlon = row[1]
    # draw great circle route
    if loanlat!=0 and loanlon!=0 and lenderlat!=0 and lenderlon!=0:
        #print "drawing", loanlat,loanlon,lenderlat,lenderlon
        m.drawgreatcircle(loanlon,loanlat,lenderlon,lenderlat,del_s=2000.0,linewidth=1,color='g',marker='3')

m.drawcoastlines()
m.fillcontinents()
# draw parallels
m.drawparallels(np.arange(10,90,20),labels=[1,1,0,1])
# draw meridians
m.drawmeridians(np.arange(-180,180,30),labels=[1,1,0,1])
ax.set_title('Location map of borrowers and lenders')
plt.show()

db.close()
```

model.py

```
import MySQLdb
import os
import pickle
from itertools import izip, chain
from collections import OrderedDict
from sklearn import linear_model
from sklearn import ensemble
from sklearn import tree
from sklearn import cross_validation
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt


db_train = MySQLdb.connect("localhost","root","password","kiva" )
db_test = MySQLdb.connect("localhost","root","password","kiva_test" )
cursor_train = db_train.cursor()
cursor_test = db_test.cursor()

train_x = []
train_y = []
test_x = []
test_y = []
sectormap =
{"Agriculture":1,"Arts":2,"Clothing":3,"Construction":4,"Education":5,"Entertainment":6,"Food":7,"H
ealth":8,"Housing":9,"Manufacturing":10,"Personal
Use":11,"Retail":12,"Services":13,"Transportation":14,"Wholesale":15}

x = []
y = []
x1 = []
y1 = []

def loadtraindata():
    # Download the file
    cursor_train.execute("select
loan_amount,paid_amount,funded_amount,disbursal_amount,sector,partner_id,gender,funded_date from
loan where rand()<=0.5")
    data = cursor_train.fetchall()
    loan_amount = 0
    paid_amount = 0
    funded_amount = 0
    disbursal_amount = 0
    for row in data:
        temp = []
        if row[0] != None:
            loan_amount = row[0]
        temp.append(loan_amount)
        if row[1] != None:
            paid_amount = row[1]
        temp.append(paid_amount)
        if row[2] != None:
            funded_amount = row[2]
        temp.append(funded_amount)
        if row[3] != None:
            disbursal_amount = row[3]
        temp.append(disbursal_amount)
        temp.append(sectormap.get(row[4]))
        temp.append(row[5])
        if row[6] == 'F':
            temp.append(1)
        else :
            temp.append(0)
        train_x.append(temp)
        funded_date = row[7]
        if funded_date == None:
            train_y.append(0)
        else :
            train_y.append(1)

    print "Number of train records:", len(train_x)
    #print train_x


def loadtestdata():
    # Download the file
    cursor_test.execute("select
loan_amount,paid_amount,funded_amount,disbursal_amount,sector,partner_id,gender,funded_date from
loan where rand()<=0.5")
    data = cursor_test.fetchall()
    loan_amount = 0
    paid_amount = 0
    funded_amount = 0
    disbursal_amount = 0

    for row in data:
        temp = []
        if row[0] != None:
            loan_amount = row[0]
        temp.append(loan_amount)
        if row[1] != None:
            paid_amount = row[1]
        temp.append(paid_amount)
        if row[2] != None:
            funded_amount = row[2]
        temp.append(funded_amount)
        if row[3] != None:
            disbursal_amount = row[3]
        temp.append(disbursal_amount)
        temp.append(sectormap.get(row[4]))
        temp.append(row[5])
        if row[6] == 'F':
            temp.append(1)
        else :
            temp.append(0)
        test_x.append(temp)
        funded_date = row[7]
        if funded_date == None:
            test_y.append(0)
        else :
            test_y.append(1)

    print "Number of test records:", len(test_x)
    #print test_x

def getProbabilities(clf,test_x):
    global problabels
    problabels = {}
    probabilities =  clf.predict_proba(test_x)
    print "length:", len(probabilities)# ,"::probabilities:", probabilities
    i = 0
    try:
        for line in probabilities:
            if len(line)==2:
                problabels[i] = line[1]
                i += 1
    except IndexError:
        print "Error when i is:",i
        print "prob:",len(problabels)
    problabels = OrderedDict(sorted(problabels.items(), key=lambda x: -x[1]))
    print "Model: ",type(clf), "No of test records:",len(problabels)


def calculateAccuracy(th,test_y):
    bullseyes = 0;
    accuracy = 0;
    if th == 0:
        th = 0.5
    for k,v in problabels.iteritems():
        #print "key, val:", k, v

        predictedLabel = 0;
        if v >= th:
            predictedLabel = 1
        #print "predictedLabel:", predictedLabel, "test_y[k]",test_y[k]
        if(test_y[k] == predictedLabel):
            bullseyes +=1;
    if len(problabels) != 0 :
        accuracy = float(bullseyes)/(len(problabels))
    print "Accuracy :"+ str(accuracy)
    return accuracy

def fitmodel():
    clflogistic = linear_model.LogisticRegression().fit(train_x, train_y)
    getProbabilities(clflogistic,train_x)
    calculateAccuracy(0,train_y)

    clfTree = tree.DecisionTreeClassifier().fit(train_x, train_y)
    getProbabilities(clfTree,train_x)
    calculateAccuracy(0,train_y)

def comparemodels():
    count = 1
    while count < 20:
        print "-----------------------------------------------------------------------"
        print "Evaluating for :" , count * 5, "%"
        print "-----------------------------------------------------------------------"
        X_train, X_test, y_train, y_test = cross_validation.train_test_split(
        train_x, train_y, test_size=count*0.05, random_state=0)
        print 'Training data#: ',len(X_train), ' Test data#: ',len(X_test)
        count += 1
        clflogistic = linear_model.LogisticRegression().fit(X_train, y_train)
        #print "Score: ", clflogistic.score(X_test,y_test)
        getProbabilities(clflogistic,X_test)
        accuracy = calculateAccuracy(0,y_test)
        x.append(count*5)
        y.append(accuracy)

        clfTree = tree.DecisionTreeClassifier().fit(X_train, y_train)
        #print "Score: ", clfTree.score(X_test,y_test)
        getProbabilities(clfTree,X_test)
        accuracy = calculateAccuracy(0,y_test)
        x1.append(count*5)
        y1.append(accuracy)
        print "-----------------------------------------------------------------------"

    plt.plot(x, y, c='r', label="Logit")
    plt.plot(x1, y1, c='g', label= "Tree")
    plt.xlabel("% Training Set") # set the x axis label
    plt.ylabel("Accuracy") # set the y axis label
    plt.legend() # place a legend on the current axes
    plt.show()

loadtraindata();
loadtestdata();
fitmodel();
comparemodels()

db_train.close()
db_test.close()
```

test.sql

```
select a.id,a.country from loan a, loan_lenders b, lender c where a.country = c.country_code limit 1000;

select a.lender_id,a.name from lender a where a.country_code is not null group by a.country_code;

select count(id),sector from loan where funded_date is not null group by sector;

select count(id),sector from loan where funded_date is null group by sector;

select count(id),country from loan where funded_date is null group by country;

select count(id),country from loan where funded_date is not null group by country;

select count(id), gender from loan where funded_date is null group by gender;

select count(id), gender from loan where funded_date is not null group by gender;

select count(lender_id) from lender where occupational_info is not null;

select count(id),activity from loan where funded_date is not null group by activity;

select count(id),activity from loan where funded_date is null group by activity;
select distinct activity,sector from loan order by sector;

select count(lender_id),country_code from lender where loan_count = 0 group by country_code;

select count(lender_id),country_code from lender where loan_count > 0 group by country_code;
```

WordCloudsample.py

```
from WordCloudMaker import WordCloudMaker
import urllib2
import MySQLdb

def getText():
    text = ""
    cursor = db.cursor()
    cursor.execute("SELECT uses from loan where uses is not null and rand() <= 0.25")
    data = cursor.fetchall()
    for row in data:
        text += row[0]
    return text

def getLenderText():
    text = ""
    cursor = db.cursor()
    cursor.execute("SELECT loan_because from lender where loan_because is not null and rand() <= 0.25")
    data = cursor.fetchall()
    for row in data:
        text += row[0]
    return text


db = MySQLdb.connect("localhost","root","password","kiva" )

client = WordCloudMaker("nq5xaswr9unlh2i12zrmx70rul1cnn", "lwftyt5nyzmorfuorqo63tsevegyjd")

text = getText()
```

```
response = client.makeWordCloud(400,text,600)
print
("_______________________________________________________________________________
_")
# now you can do something with the response.
print response.body
print response.body["url"]

text = getLenderText()
response = client.makeWordCloud(400,text,600)
print
("_______________________________________________________________________________
_")
# now you can do something with the response.
print response.body
print response.body["url"]
```

kiva.py

```
## CODE TO COUNT THE NUMBER OF DEFAULTED, PAID, DELETED, ETC... LOANS
try: import simplejson as json
except ImportError: import json
status =[]
x= '1' # Open's numbered files
y = 1 # Increment's number to open file
while y < 1016:
rawData = open( x + '.json','r')
loans = {}
for row in rawData:
data = json.loads(row)
loans = data['loans']
for row in loans:
status.append(row['status']) # Pulls out status from Loan Data
print y
y = y+1
x = str(y)
#COUNT STATUS
```

```
i = 0
paid =0
deleted = 0
defaulted = 0
refunded = 0
in_repayment = 0
inactive_expired = 0
fundraising = 0
expired = 0
while i< len(status):
if status[i] == "deleted":
deleted = deleted = deleted +1
elif status[i] == "paid":
paid= paid+1
elif status[i] == "defaulted":
defaulted= defaulted+1
elif status[i] == "in_repayment":
in_repayment = in_repayment + 1
elif status[i] == "inactive_expired":
inactive_expired = inactive_expired + 1
elif status[i] == "fundraising":
fundraising = fundraising+1
elif status[i] == "expired":
expired = expired+1
i=i+1
print "the number of loans in default is equal to:", defaulted
print "the number of loans paid is equal to:", paid
print "the number of loans deleted is equal to:", deleted
print "the number of loans in repayment is equal to:", in_repayment
print "the number of loans inactive/expired is equal to:", inactive_expired
print "the number of loans fundraising is equal to:", fundraising
```

kiva_next.py

```
from operator import itemgetter
try: import simplejson as json
except ImportError: import json
```

```
activities = []
id = []
status =[]
funded_amount = []
sector = []
country = []
country_code = []
town = []
x= '1' #Pulls in all files one by one 1 through 1015
y = 1
while y < 1016:
rawData = open( x + '.json','r')
loans = {}
for row in rawData:
data = json.loads(row)
loans = data['loans']
for row in loans: #pull out data needed
sector.append(row['sector'])
activities.append(row['activity'])
id.append(row['id'])
funded_amount.append(row['funded_amount'])
status.append(row['status'])
location = (row['location']) #pull out location data
town.append(location['town'])
country_code.append(location['country_code'])
country.append(location['country'])
print y #allows us to tell the code is working away
y = y+1
x = str(y)
defaulted_sector = []
defaulted_country_code =[]
defaulted_countries =[]
defaulted_town = []
defaulted_activity = []
defaulted_amount = []
defaulted = 0
```

```
print "the length of status is:", len(status)
i = 0
while i< len(status): #record stats about entries that have defaulted
if status[i] == "defaulted":
defaulted = defaulted+1
defaulted_sector.append(sector[i])
defaulted_countries.append(country[i])
defaulted_country_code.append(country_code[i])
defaulted_activity.append(activities[i])
defaulted_amount.append(funded_amount[i])
defaulted_town.append(town[i])
i=i+1
print len(defaulted_town)
print "complete"
j = 0
r=0
countries = []
countries_count = [] #Figure out how many loan defaults each country has
while j< len(defaulted_countries):
if defaulted_countries[j] not in countries:
countries.append(defaulted_countries[j])
countries_count.append(defaulted_countries.count(defaulted_countries[j]))
elif defaulted_countries[j] in countries:
r=r+1
j=j+1
x= 0
dict ={}
cumulative_count = []
while x<len(countries): # organize the data in a more convienient form
#print "the country:", countries[x], "has the following number of loans
defaulted:", countries_count[x]
dict["country"] = countries[x]
dict["number"] = countries_count[x]
cumulative_count.append(dict)
dict ={}
x=x+1
```

```
sorted_countries_list = sorted(cumulative_count, key=itemgetter('number'))
sorted_countries_list.reverse()
print sorted_countries_list
#It will be more valuable to look at how many loans were made in each country
compared the number that have defaulted
# country [] defined above
total_loans_country = []
total_loans_country_count = []
m = 0
while m<len(country): # organize the data in a more convienient form
if country[m] not in total_loans_country:
total_loans_country.append(country[m])
total_loans_country_count.append(country.count(country[m]))
m=m+1
x= 0
dict ={}
total_cumulative_count_country = []
while x<len(total_loans_country): # organize the data in a more convienient
form
dict["country"] = total_loans_country[x]
dict["number"] = total_loans_country_count[x]
total_cumulative_count_country.append(dict)
dict ={}
x=x+1
print total_cumulative_count_country
print "part 2 complete"
final_country_percentage = []
dict = {}
x=0
while x<len(sorted_countries_list):
y=0
while y<len(total_cumulative_count_country):
if sorted_countries_list[x]['country'] == total_cumulative_count_country
[y]['country']:
print sorted_countries_list[x]['country']
print float((float(sorted_countries_list[x]['number']))*100/float
((total_cumulative_count_country[y]['number'])))
dict['country']= sorted_countries_list[x]['country']
dict['percentage'] = float((float(sorted_countries_list[x]['number'])
)*100/float((total_cumulative_count_country[y]['number'])))
final_country_percentage.append(dict)
dict={}
y=y+1
x=x+1
final_sorted_countries_list = sorted(final_country_percentage, key=itemgetter
('percentage'))
final_sorted_countries_list.reverse()
print final_sorted_countries_list
```

Python Code to load the data into MySQL tables

Load_loan.py

```
import MySQLdb
import lender
import json
import sys
import os
import random

dirname = "C:\\Documents and
Settings\\rangas1\\Desktop\\Stern\\PracticalDataScience\\project\\data\\loans\\"

#filename = "C:\\Documents and
Settings\\rangas1\\Desktop\\Stern\\PracticalDataScience\\project\\data\\loans\\1.json"
data = []

def readfile(filename):
  try :
    filehandle = open(dirname+filename,"r")
  except IOError :
    print "Data file read error. Please check whether data file was downloaded and stored properly."
```

```
        raise
    rawdata = json.load(filehandle)
    #print str(data["lenders"])
    global data
    data = rawdata["loans"]
    print len(data)


def insertrecord(data):

    db_train = MySQLdb.connect("localhost","root","password","kiva" )
    db_test = MySQLdb.connect("localhost","root","password","kiva_test" )
    db_train.autocommit(True)
    db_test.autocommit(True)
    cursor_train = db_train.cursor()
    cursor_test = db_test.cursor()

    for line in data:
        loan_id = line["id"]
        print loan_id
        name = line["name"]
        status = line["status"]
        funded_amount = line["funded_amount"]
        paid_amount = line["paid_amount"]
        image_id = line["image"]["id"]
        template_id = line["image"]["template_id"]
        activity = line["activity"]
        sector = line["sector"]
        uses = line["use"]
        country_code = line["location"]["country_code"]
        town = line["location"]["town"]
        geolevel = line["location"]["geo"]["level"]
        geopairs = line["location"]["geo"]["pairs"]
        geotype = line["location"]["geo"]["type"]
        partner_id = line["partner_id"]
        disbursal_amount = line["terms"]["disbursal_amount"]
        disbursal_currency = line["terms"]["disbursal_currency"]
        disbursal_date = line["terms"]["disbursal_date"]
        if disbursal_date != None:
            disbursal_date = disbursal_date[:10]
        loan_amount = line["terms"]["loan_amount"]
        nonpayment = line["terms"]["loss_liability"]["nonpayment"]
        currency_exchange = line["terms"]["loss_liability"]["currency_exchange"]
        posted_date = line["posted_date"]
        if posted_date!= None:
            posted_date = posted_date[:10]
        funded_date = line["funded_date"]
        if funded_date != None:
            funded_date = funded_date[:10]
        paid_date = line["paid_date"]
        if paid_date != None:
            paid_date = paid_date[:10]
        loan_amount = line["loan_amount"]
        journal_entries = line["journal_totals"]["entries"]
        journal_bulk_entries = line["journal_totals"]["bulkEntries"]
        if len(line["borrowers"]) == 1:
            gender = line["borrowers"][0]["gender"]
        elif len(line["borrowers"]) > 1:
            gender = 'N'

        try:
            if(random.random() > 0.7):
                cursor_test.execute("""INSERT INTO loan values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,date(%s),date(%
s),%s,date(%s),%s,%s,%s,%s)""",(loan_id,name ,status,funded_amount
,paid_amount,image_id,template_id, activity,sector,uses,country_code,town ,geolevel,geopairs,geotype
,partner_id
,disbursal_amount,disbursal_currency,disbursal_date,loan_amount,nonpayment,currency_exchange
,posted_date,funded_date,None,paid_date,None,journal_entries, journal_bulk_entries,gender))
                db_test.commit()
            else:
                cursor_train.execute("""INSERT INTO loan values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,date(%s),date(%
s),%s,date(%s),%s,%s,%s,%s)""",(loan_id,name ,status,funded_amount
,paid_amount,image_id,template_id, activity,sector,uses,country_code,town ,geolevel,geopairs,geotype
,partner_id
,disbursal_amount,disbursal_currency,disbursal_date,loan_amount,nonpayment,currency_exchange
,posted_date,funded_date,None,paid_date,None,journal_entries, journal_bulk_entries,gender))
                db_train.commit()
        except UnicodeEncodeError:
            print 'Unicode char', id
        except NameError as e:
            print "*********************************Rolling back************"
            print sys.exc_info()[0]
            print e
            db.rollback()
        except :
            print "Error:", sys.exc_info()[0]

    db_test.close()
    db_train.close()

for filename in os.listdir(dirname):
    print  filename
    readfile(filename)
    insertrecord(data)
```

load_loan_api.py

```
import os
import httplib2
import json
import MySQLdb
import sys
import random

db_train = MySQLdb.connect("localhost","root","password","kiva" )
db_test = MySQLdb.connect("localhost","root","password","kiva_test" )
```

```
db_train.autocommit(True)
db_test.autocommit(True)
cursor_train = db_train.cursor()
cursor_test = db_test.cursor()

app_idstr = "&app_id=edu.stern.nyu.pds-f2012"
h = httplib2.Http()
for count in range(70,400):
    resp, content = h.request("http://api.kivaws.org/v1/loans/newest.json?page="+str(count)+app_idstr)
    assert resp.status == 200
    print resp
    print content
    data = json.loads(content)
    print len(data)
    print len(data["loans"])
    ids = ""
    count = 0
    for lender in data["loans"]:
        count += 1
        loan_id = str(lender["id"])
        if count != 10:
            ids += loan_id +","
        else:
            ids += loan_id
            break
    app_idstring = "?app_id=edu.stern.nyu.pds-f2012"
    requeststr = "http://api.kivaws.org/v1/loans/"+ids+".json"+app_idstring
    print requeststr
    resp, content = h.request(requeststr)
    data = json.loads(content)
    print content
    print data["loans"]
    print type(data["loans"])

    for line in data["loans"]:
        funded_date = ""
```

```
paid_date = ""
loan_id = line["id"]
print loan_id
name = line["name"]
status = line["status"]
funded_amount = line["funded_amount"]
paid_amount = None
if line.has_key("paid_amount"):
    paid_amount = line["paid_amount"]
image_id = line["image"]["id"]
template_id = line["image"]["template_id"]
activity = line["activity"]
sector = line["sector"]
uses = line["use"]
country_code = line["location"]["country_code"]
town = None
if line["location"].has_key("town"):
    town = line["location"]["town"]
geolevel = line["location"]["geo"]["level"]
geopairs = line["location"]["geo"]["pairs"]
geotype = line["location"]["geo"]["type"]
partner_id = line["partner_id"]
disbursal_amount = line["terms"]["disbursal_amount"]
disbursal_currency = line["terms"]["disbursal_currency"]
disbursal_date = line["terms"]["disbursal_date"]
disbursal_date = None
if disbursal_date != None:
    disbursal_date = disbursal_date[:10]
loan_amount = line["terms"]["loan_amount"]
nonpayment = line["terms"]["loss_liability"]["nonpayment"]
currency_exchange = line["terms"]["loss_liability"]["currency_exchange"]
posted_date = line["posted_date"]
posted_date = None
if posted_date!= None:
    posted_date = posted_date[:10]
funded_date = None
```

```
        paid_date = None
        if line.has_key("funded_date"):
            funded_date = line["funded_date"]
        if funded_date != None:
            funded_date = funded_date[:10]
        if line.has_key("paid_date"):
            paid_date = line["paid_date"]
        if paid_date != None:
            paid_date = paid_date[:10]
        loan_amount = line["loan_amount"]
        journal_entries = line["journal_totals"]["entries"]
        journal_bulk_entries = line["journal_totals"]["bulkEntries"]
        if len(line["borrowers"]) == 1:
            gender = line["borrowers"][0]["gender"]
        elif len(line["borrowers"]) > 1:
            gender = 'N'

        try:
            if(random.random() > 0.7):
                cursor_test.execute("""INSERT INTO loan values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,date(%s),date(%
s),%s,date(%s),%s,%s,%s,%s)""",(loan_id,name ,status,funded_amount
,paid_amount,image_id,template_id, activity,sector,uses,country_code,town ,geolevel,geopairs,geotype
,partner_id
,disbursal_amount,disbursal_currency,disbursal_date,loan_amount,nonpayment,currency_exchange
,posted_date,funded_date,None,paid_date,None,journal_entries, journal_bulk_entries,gender))
                db_test.commit()
            else:
                cursor_test.execute("""INSERT INTO loan values
(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,date(%s),date(%
s),%s,date(%s),%s,%s,%s,%s)""",(loan_id,name ,status,funded_amount
,paid_amount,image_id,template_id, activity,sector,uses,country_code,town ,geolevel,geopairs,geotype
,partner_id
,disbursal_amount,disbursal_currency,disbursal_date,loan_amount,nonpayment,currency_exchange
,posted_date,funded_date,None,paid_date,None,journal_entries, journal_bulk_entries,gender))
                db_train.commit()
        except UnicodeEncodeError:
            print 'Unicode char', loan_id
        except NameError as e:
            print "********************************Rolling back************"
            print sys.exc_info()[0]
            print e
            db_test.rollback()
            db_train.rollback()
        except :
            continue

db_test.close()
db_train.close()
```

load_loan_lenders.py

```
import MySQLdb
import httplib2
import json
import _mysql_exceptions

db = MySQLdb.connect("localhost","root","password","kiva" )

cursor = db.cursor()

cursor.execute("SELECT id from loan")
data = cursor.fetchall()
app_idstr = "&app_id=edu.stern.nyu.pds-f2012"
h = httplib2.Http()

for row in data:
    print "id: %s " % row[0]
    loan_id = str(row[0])
    resp, content = h.request("http://api.kivaws.org/v1/loans/"+loan_id+"/lenders.json")
    print resp
    print content
    data = json.loads(content)
    print len(data)
    print len(data["lenders"])
    for lender in data["lenders"]:
        if lender.has_key("lender_id"):
            lender_id = lender["lender_id"]
            print "lender_id ", lender_id
            try:
                cursor.execute("""INSERT INTO LOAN_LENDERS values(%s,%s)""",(loan_id,lender_id))
                db.commit()
            except :
                print "foriegn key violation, continuing"
db.close()
```

load_lender.py

```
import MySQLdb
import lender
import json
import sys
import os
import random

dirname = "C:\\Documents and
Settings\\rangas1\\Desktop\\Stern\\PracticalDataScience\\project\\data\\lenders\\"

#filename = "C:\\Documents and
Settings\\rangas1\\Desktop\\Stern\\PracticalDataScience\\project\\data\\lenders\\1.json"
data = []

def readfile(filename):
    try :
        filehandle = open(dirname+filename,"r")
    except IOError :
        print "Data file read error. Please check whether data file was downloaded and stored properly."
        raise
    rawdata = json.load(filehandle)
    #print str(data["lenders"])
    global data
    data = rawdata["lenders"]
    print len(data)



def insertrecord(data):
    db_train = MySQLdb.connect("localhost","root","password","kiva" )
    db_test = MySQLdb.connect("localhost","root","password","kiva_test" )
    db_train.autocommit(True)
    db_test.autocommit(True)
    cursor_train = db_train.cursor()
    cursor_test = db_test.cursor()

    print type(data)
    for line in data:
        lender_id = line["lender_id"]
        print lender_id
        if lender_id == None:
            continue;
        name = line["name"]
        image_id = line["image"]["id"]
        template_id = line["image"]["template_id"]
        whereabouts = line["whereabouts"]
        country_code = line["country_code"]
        uid = line["uid"]
        member_since = line["member_since"]
        personal_url = line["personal_url"]
        occupation = line["occupation"]
        loan_because = line["loan_because"]
        occupational_info = line["occupational_info"]
        loan_count = line["loan_count"]
        invitee_count = line["invitee_count"]
        inviter_id = line["inviter_id"]

        try:
            #cursor.execute("""INSERT INTO lender values
('%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s')""",(lender_id,name,image_i
d,template_id,whereabouts,country_code,lender_uid,member_since,personal_url,occupation,loan_because,o
ccupational_info,loan_count,invitee_count))
            if(random.random() > 0.7):
                cursor_test.execute("""INSERT INTO lender values
(%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,%s,%s,%s)""",(lender_id,name,image_id,template_id,whe
reabouts,country_code,uid,member_since[:10],personal_url,occupation,loan_because,occupational_info,loa
n_count,invitee_count))
                db_test.commit()
            else:
                cursor_train.execute("""INSERT INTO lender values
(%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,%s,%s,%s)""",(lender_id,name,image_id,template_id,whe
reabouts,country_code,uid,member_since[:10],personal_url,occupation,loan_because,occupational_info,loa
n_count,invitee_count))
                db_train.commit()
        except UnicodeEncodeError:
            print 'Unicode char', lender_id
        except NameError as e:
            print "*********************************Rolling back************"
            print sys.exc_info()[0]
            print e
            db_test.rollback()
            db_train.rollback()

    db_test.close()
    db_train.close()

for filename in os.listdir(dirname):
    print  filename
    readfile(filename)
insertrecord(data)
```

load_lender_api.py

```
import os
import httplib2
import json
import MySQLdb
import sys
import random

db_train = MySQLdb.connect("localhost","root","password","kiva" )
db_test = MySQLdb.connect("localhost","root","password","kiva_test" )
db_train.autocommit(True)
db_test.autocommit(True)
cursor_train = db_train.cursor()
cursor_test = db_test.cursor()

app_idstr = "&app_id=edu.stern.nyu.pds-f2012"
h = httplib2.Http()
for count in range(2,500):
    resp, content = h.request("http://api.kivaws.org/v1/lenders/newest.json?page="+str(count)+app_idstr)
    assert resp.status == 200
    print resp
    print content
    data = json.loads(content)
    print len(data)
    print len(data["lenders"])
    ids = ""
    count = 0
    for lender in data["lenders"]:
        count += 1
        lender_id = lender["lender_id"]
        if count != 50:
            ids += lender_id +","
        else:
            ids += lender_id
    app_idstring = "?app_id=edu.stern.nyu.pds-f2012"
    requeststr = "http://api.kivaws.org/v1/lenders/"+ids+".json"+app_idstring
    print requeststr
    resp, content = h.request(requeststr)
    data = json.loads(content)
    print data["lenders"]
    print type(data["lenders"])
    for line in data["lenders"]:
        lender_id = line["lender_id"]
        print lender_id
        if line.has_key("name"):
            name = line["name"]
        else :
            name = None
        if line.has_key("image") :
            image_id = line["image"]["id"]
        else :
            image_id = ""

        template_id = line["image"]["template_id"]
        whereabouts = line["whereabouts"]
        if line.has_key("country_code") :
            country_code = line["country_code"]
        else:
            country_code = None

        uid = line["uid"]
        member_since = line["member_since"]
        personal_url = line["personal_url"]
        occupation = line["occupation"]
        loan_because = line["loan_because"]
        occupational_info = line["occupational_info"]
        loan_count = line["loan_count"]
        invitee_count = line["invitee_count"]
        if line.has_key("inviter_id") :
            inviter_id = line["inviter_id"]
        else:
            inviter_id = None

        try:
            #cursor.execute("""INSERT INTO lender values
('%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s','%s')""",(lender_id,name,image_i
d,template_id,whereabouts,country_code,lender_uid,member_since,personal_url,occupation,loan_because,o
ccupational_info,loan_count,invitee_count))
      if(random.random() > 0.7):
        cursor_test.execute("""INSERT INTO lender values
(%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,%s,%s,%s)""",(lender_id,name,image_id,template_id,whe
reabouts,country_code,uid,member_since[:10],personal_url,occupation,loan_because,occupational_info,loa
n_count,invitee_count))
        #print
lender_id,name,image_id,template_id,whereabouts,country_code,uid,member_since[:10],personal_url,occu
pation,loan_because,occupational_info,loan_count,invitee_count
        db_test.commit()
      else:
        cursor_train.execute("""INSERT INTO lender values
(%s,%s,%s,%s,%s,%s,%s,date(%s),%s,%s,%s,%s,%s,%s)""",(lender_id,name,image_id,template_id,whe
reabouts,country_code,uid,member_since[:10],personal_url,occupation,loan_because,occupational_info,loa
n_count,invitee_count))
        #print
lender_id,name,image_id,template_id,whereabouts,country_code,uid,member_since[:10],personal_url,occu
pation,loan_because,occupational_info,loan_count,invitee_count
        db_train.commit()
    except UnicodeEncodeError:
      print 'Unicode char', lender_id
    except NameError as e:
      print "********************************Rolling back************"
      print sys.exc_info()[0]
      print e
      db_test.rollback()
      db_train.rollback()

db_test.close()
db_train.close()
```

load_lender_loans.py

```
import MySQLdb
import httplib2
import json
```

```
    import _mysql_exceptions

    db = MySQLdb.connect("localhost","root","password","kiva" )

    cursor = db.cursor()

    cursor.execute("SELECT lender_id from lender")
    data = cursor.fetchall()
    app_idstr = "?app_id=edu.stern.nyu.pds-f2012"
    h = httplib2.Http()

    for row in data:
        print "id: %s " % row[0]
        lender_id = str(row[0])
        resp, content = h.request("http://api.kivaws.org/v1/lenders/"+lender_id+"/loans.json"+app_idstr)
        if resp.status != 200:
            continue;
        print resp
        print content
        data = json.loads(content)
        print len(data)
        print len(data["loans"])
        for loan in data["loans"]:
            if loan.has_key("id"):
                loan_id = loan["id"]
                print "Loan id ", loan_id
                try:
                    cursor.execute("""INSERT INTO LENDER_LOANS values(%s,%s)""",(lender_id,loan_id))
                    db.commit()
                except :
                    print "foriegn key violation, continuing"
db.close()
```