

# Advanced Data Science & Architecture

# Loan Data Analysis

- *Under the guidance of Sri Krishnamurthy*

Compiled By,

Mohit Mittal  
Sneha Ravikumar  
Taj Poovaiah

## CONTENTS

About Application: .....	3
STEPS TO RUN THE Pipeline.....	3
DOCKER ON AMAZON AWS SCHEDULING: .....	4
Downloading Data: .....	7
Cleaning Data: LOAN DATA SET .....	8
SUMMARY OBSERVATIONS AFTER DOWNLOAD .....	8
SUMMARY OBSERVATIONS POST CLEANING.....	11
Handling Missing Data: LOAN DATA .....	13
CLEANING: Declined Loan Data Set .....	15
Summary Observations made after replacing the missing values of the risk score with median and mean. Replaced the missing values with 999 .....	16
Microsoft Azure ML STUDIO to cross validate our findings and visualize Cleaning, Summary, and Feature Engineering .....	17
Checking for missing values using Summarize Data using Microsoft Azure ML Studio .....	17
Cleaning data and checking for missing values using Microsoft Azure ML Studio .....	18
Exploratory Data Analysis: LOAN DATA .....	19
USING JUPYTER NOTEBOOK AND R <b>Published Link</b> : <a href="http://rpubs.com/Palecanda">http://rpubs.com/Palecanda</a> .....	19
ANALYSIS for Loan Data: GRADES .....	19
LOAN DATA SET GRADE FREQUENCY.....	19
Exploring interest rates based on Grades assigned by Lending Club .....	20
Paid Vs. Unpaid loan amount over the Grades.....	21
INTEREST RATES AGAINST GRADES .....	22
ANALYSIS for Loan Data: LOAN AMOUNT.....	23
LOAN AMOUNT AGAINST LOAN STATUS.....	23
COUNT OF LOAN AMOUNT AGAINST GRADE .....	24
EXPLORATORY ANALYSIS: DECLINED LOAN DATA .....	25
<b>REJECTED LOAN AMOUNT AGAINST EMPLOYMENT LENGTH VS RISK SCORE</b> .....	25
COUNT OF ACCEPTED AND REJECTED LOAN AMOUNT AGAINST STATES .....	27
Totals Funded By State .....	29
Feature Engineering: PART 1 .....	31

<b>Deployment</b> .....	33
<b>Classification</b> .....	34
<b>ROC CURVES</b> .....	35
<b>FEATURE SELECTION FOR CLUSTERING USING R</b> .....	37
<b>Clustering</b> .....	38
<b>Clustering using k-means for numeric and categorical values</b> .....	38
<b>MANUAL CLUSTERING USING PYTHON</b> .....	40
<b>PREDICTION</b> .....	42
PREDICTION ON THE ENTIRE DATASET .....	42
PREDICTION MODELS USING MICROSOFT AZURE .....	43
<b>Web Application: HOW IT WORKS</b> .....	45

## ABOUT APPLICATION:

**Programming Language used : Python**

**Workflow Manager User: Luigi**

### Tasks

- A. Downloading Loan Data (GetData.py)**
- B. Clean Loan Data (CleanData.py)**
- C. Upload Preprocessed data to Amazon S3 (LoanData.py)**

## STEPS TO RUN THE PIPELINE

1. Pull the Docker image (mohit914/test:LCv1.05) from the dockerhub  
**docker pull mohit914/test:LCv1.05**
2. Run the docker image. It will take you into the bash terminal  
**docker run -ti mohit914/test:LCv1.05**
3. Inside the bash terminal run the python application by using the following command  
**python LoanData.py Start --local-scheduler**  
**python RejectLoanData.py Start --local-scheduler**  
This program will run the following tasks automatically:
  - i. **GetData()** – It will download the LendingClub.com loan dataset
  - ii. **ClenaData()** – It will clean and preprocess the LendingClub loan dataset
  - iii. **Start()** – It will upload the preprocessed data to Amazon S3 Bucket. ***You will need to provide your Amazon Access key and Secret Key to complete this task.***

**Docker Trouboulshooting:**

1. If running docker gives No space on the device error, then remove all the images and follow the steps to run the pipeline again. If problem still exists, remove all the images and containers on your device and follow the steps to run the pipeline again.
2. If the problem still exists, please email us to let us know the problem so that we can try to troubleshoot the problem.

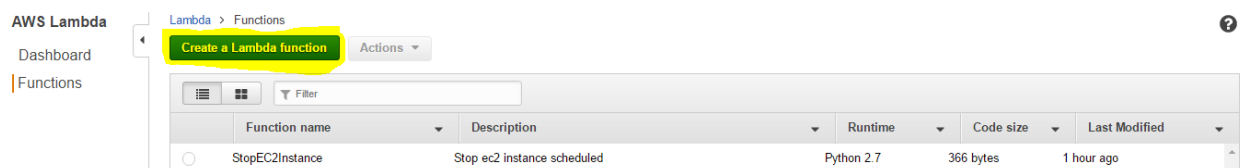
## DOCKER ON AMAZON AWS SCHEDULING:

### Docker on AWS – Scheduling

1. Connect to Amazon Linux AMI instance
2. Update the installed package cache on the instance  
**sudo yum update -y**
3. Install Docker  
**sudo yum install -y docker**
4. Start the docker service  
**sudo service docker start**
5. Add the ec2-user to the docker group so that you can execute Docker commands without using **sudo**  
**usermod -a -G docker ec2-user**
6. Log out and log back in again to pick up the new docker group permissions
7. Pull the docker image from dockerhub  
**docker pull mohit914/test:LCv1.05**

### To schedule the task to start and stop the ec2 instance

1. Create a Lambda function by going on the AWS Lambda console and clicking on Create Lambda function.



2. Choose Configure triggers and then choose next

Lambda > New function

Select blueprint


**Configure triggers**

Configure function

Review

### Configure triggers

You can choose to add a trigger that will invoke your function.



Lambda
Remove

Cancel Previous Next

3. Enter the values for name and description of the function and select python 2.7.

Services Resource Groups

Lambda > New function

Select blueprint

Configure triggers

**Configure function**

Review

### Configure function

A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.

Name\* StartEC2Instances

Description Starts ec2 instance at the scheduled time

Runtime\* Python 2.7

4. Enter the code to start the instance in the “Lambda function code” section below. Replace the highlighted values with your region and instance id respectively.

Runtime\* Python 2.7

### Lambda function code

Provide the code for your function. Use the editor if your code does not require custom libraries (other than boto3). If you need custom libraries, you can upload your code and libraries as a .ZIP file.

Code entry type Edit code inline

```

1 import boto3
2
3 # Enter the region your instances are in, e.g. 'us-east-1'
4
5 region = 'XX-XXXXX-X'
6
7 # Enter your instances here: ex. ['X-XXXXXXXXX', 'X-XXXXXXXXX']
8
9 instances = ['X-XXXXXXXXX']
10
11
12
13 def lambda_handler(event, context):
14
15     ec2 = boto3.client('ec2', region_name=region)
16
17     ec2.stop_instances(InstanceIds=instances)
18
19     print 'started your instances: ' + str(instances)
20
21
22

```

5. Select the role (create if it doesn't exist to start and stop the ec2 instance)

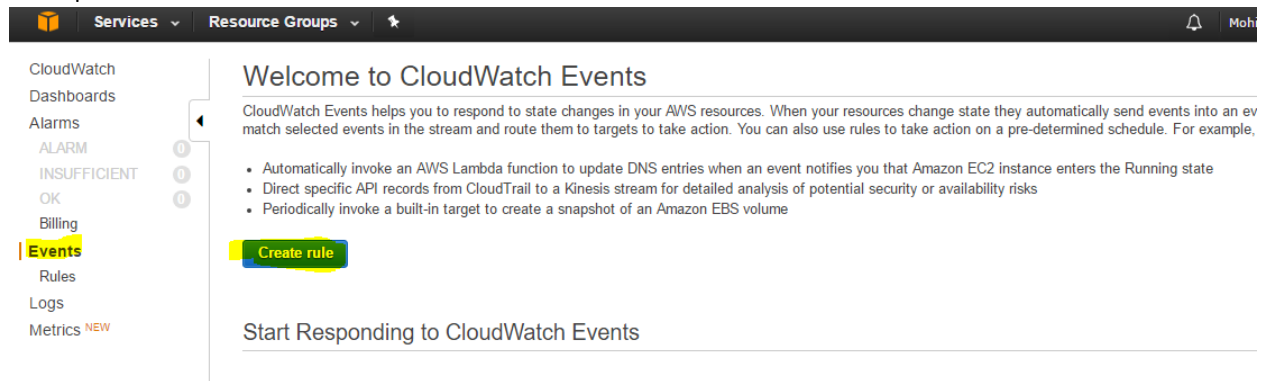
### Lambda function handler and role

Handler\*  ⓘ

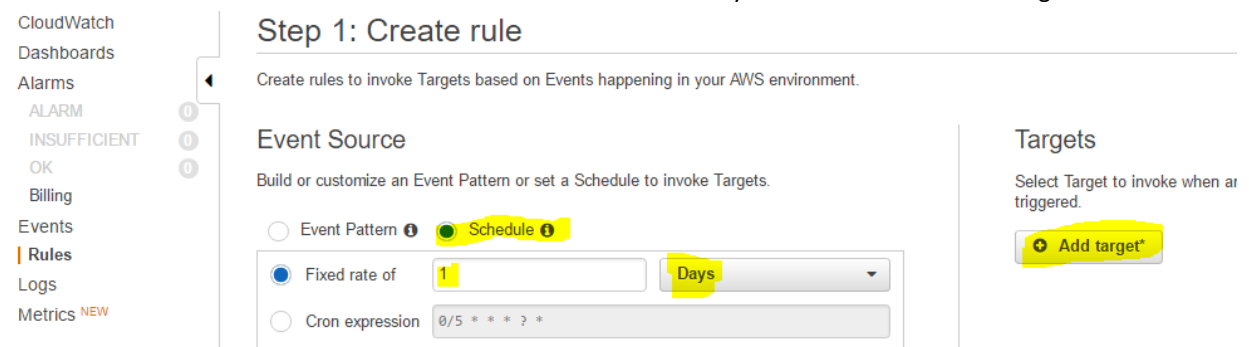
Role\*  ⓘ

Existing role\*  ⓘ

6. Click next and click on Create function
7. Open CloudWatch console and select Events.



8. Select Schedule under Event Source and select fixed rate of 1 Days. And then click on add target.



9. Select Lambda Functions and the function name.

## Targets

Select Target to invoke when an event matches your Event Pattern or when schedule is triggered.

Lambda function

Function\* StartEC2Instances

- Configure version/alias
- Configure input

+ Add target\*

10. Click on configure details. And enter the name and description for the event.

Step 2: Configure rule details

Rule definition

Name\* Start\_instance\_event

Description event starts ec2 instance at scheduled time

State ☒ Enabled

CloudWatch Events will add necessary permissions for target(s) so they can be invoked when this rule is triggered.

\* Required

Cancel Back Create rule

11. Follow the same steps to create an event to stop the instance on scheduled time.

## DOWNLOADING DATA:

File Location: Classes/LoanData/GetData.py

Task Requires no prior tasks to be completed.

Output of the task are all the Loan Data files.

Process:

- Asking user for username and password.
- Creating a browser agent (using the mechanicalsoup library) to store and pass the cookies
- Logging in with the user's credentials
- Checking if the user is successfully logged in or not.
- Landing to the page that contains the list of files and download links
- Putting the table of files in a dataframe
- Iterating through the rows in dataframe for the links that contain sample files and downloading them to a newly created (if it doesn't already exist) "Downloads" directory
- The program also checks if the files are already present in the "Downloads" directory. It skips the downloading if the file already exists.
- Unzipping the downloaded file.

The Downloaded data can be found in the directory : "Data/Downloads/"

## CLEANING DATA: LOAN DATA SET

### SUMMARY OBSERVATIONS AFTER DOWNLOAD

We obtained the summaries for Loan data after download and after cleaning. We see that the count of missing values have been changed. All the columns after cleaning have the same number of values as seen by the count value. The details of how we are handling missing data in each column is given after "summary of data after cleaning"



	count	mean	std	min	25%	50%	75%	max
member_id	0							
loan_amnt	1321847	14748.22	8622.143	500	8000	12900	20000	40000
funded_amnt	1321847	14739.22	8618.488	500	8000	12850	20000	40000
funded_amnt_inv	1321847	14710.71	8625.729	0	8000	12800	20000	40000
installment	1321847	439.1395	253.2311	14.01	255.9	380.25	578.31	1584.9
annual_inc	1321843	76495.94	69111.92	0	46000	65000	90000	9573072
url	0							
dti	1321847	18.87239	70.98519	-1	12.04	17.83	24.24	9999
delinq_2yrs	1321818	0.329594	0.892043	0	0	0	0	39
inq_last_6mths	1321817	0.650875	0.957663	0	0	0	1	33
mths_since_last_delinq	662677	33.89481	21.89243	0	15	31	49	195
mths_since_last_record	219977	68.9857	26.86579	0	51	70	88	129
open_acc	1321818	11.6559	5.46941	0	8	11	14	97
pub_rec	1321818	0.215028	0.617163	0	0	0	0	86
revol_bal	1321847	16928.01	22933.9	0	6295	11652	20584	2904836
total_acc	1321818	25.02217	11.90355	1	16	23	32	176
out_prncp	1321847	5971.41	7531.106	0	0	2927.55	9990.595	40205.27
out_prncp_inv	1321847	5968.963	7528.321	0	0	2926.1	9986.6	40205.27
total_pymnt	1321847	10065.93	8677.782	0	3677.26	7498.63	13686.11	61501.88
total_pymnt_inv	1321847	10038.57	8667.587	0	3660.27	7472.47	13647.27	61414.02
total_rec_prncp	1321847	7790.732	7367.401	0	2408.46	5319.5	10573.48	40000
total_rec_int	1321847	2192.841	2367.291	0	689.14	1430.24	2798.65	26501.88
total_rec_late_fee	1321847	0.629722	5.518013	0	0	0	0	437.99
recoveries	1321847	81.7311	533.0982	0	0	0	0	39444.37
collection_recovery_fee	1321847	12.08557	89.39024	0	0	0	0	7002.19
last_pymnt_amnt	1321847	2840.73	5556.066	0	298.14	505.6	1327.635	42148.53
collections_12_mths_ex_med	1321702	0.016785	0.146722	0	0	0	0	20
mths_since_last_major_derog	347331	44.41145	22.28005	0	27	44	62	197
policy_code	1321847	1	0	1	1	1	1	1
annual_inc_joint	9300	111563.6	50199.77	11000	78492.5	103000	134000	1050000
dti_joint	9296	18.29942	7.020419	0.32	13.32	18.01	22.9	69.49
acc_now_delinq	1321818	0.005582	0.081198	0	0	0	0	14
tot_coll_amnt	1251571	241.1676	8444.208	0	0	0	0	9152545
tot_cur_bal	1251571	140842.8	156363.7	0	30017	80847	209787.5	8000078
open_acc_6m	455717	1.008288	1.188975	0	0	1	2	18
open_il_6m	455718	2.831942	3.052855	0	1	2	3	48
open_il_12m	455718	0.727941	0.978708	0	0	0	1	25
open_il_24m	455718	1.629345	1.657855	0	0	1	2	51
mths_since_rcnt_il	443459	21.27134	26.62149	0	7	13	24	511
total_bal_il	455718	35619.87	42570.6	0	9495	23781	46364	1547285
il_util	395615	70.98847	23.17377	0	58	74	87	1000
open_rv_12m	455718	1.373542	1.536616	0	0	1	2	28

open_rv_24m	455718	2.903416	2.631446	0	1	2	4	60
max_bal_bc	455718	5777.328	5619.377	0	2362	4385	7460	776843
all_util	455695	60.22548	20.15962	0	47	61	74	204
total_rev_hi_lim	1251571	32764.69	36690.14	0	14100	24100	40600	9999999
inq_fi	455718	0.965009	1.497837	0	0	0	1	48
total_cu_tl	455717	1.498709	2.722067	0	0	0	2	111
inq_last_12m	455717	2.166066	2.45905	0	0	1	3	49
acc_open_past_24mths	1271817	4.551259	3.120103	0	2	4	6	64
avg_cur_bal	1251559	13365.83	16004.96	0	3149	7424	18518	958084
bc_open_to_buy	1259244	9547.611	14582.59	0	1345	4359	11520	711140
bc_util	1258527	62.10613	27.51823	0	41.6	65.8	86.2	339.6
chargeoff_within_12_mths	1321702	0.008963	0.10839	0	0	0	0	10
delinq_amnt	1321818	15.61729	812.8777	0	0	0	0	185408
mo_sin_old_il_acct	1214312	127.0696	52.01166	0	100	130	153	724
mo_sin_old_rev_tl_op	1251570	184.3689	94.84415	2	119	167	234	901
mo_sin_rcnt_rev_tl_op	1251570	13.44386	16.80114	0	4	8	16	438
mo_sin_rcnt_tl	1251571	8.086934	9.06588	0	3	6	10	314
mort_acc	1271817	1.689627	2.013544	0	0	1	3	61
mths_since_recent_bc	1260066	24.61952	31.49733	0	6	14	29	639
mths_since_recent_bc_dlq	318765	39.41078	22.75385	0	20	38	58	195
mths_since_recent_inq	1139670	6.833985	5.923765	0	2	5	10	25
mths_since_recent_revol_delinq	448529	35.46095	22.41687	0	17	32	52	197
num_accts_ever_120_pd	1251571	0.505305	1.309245	0	0	0	0	51
num_actv_bc_tl	1251571	3.714223	2.264803	0	2	3	5	47
num_actv_rev_tl	1251571	5.778005	3.334688	0	3	5	7	59
num_bc_sats	1263257	4.745405	2.942177	0	3	4	6	71
num_bc_tl	1251571	8.150501	4.788671	0	5	7	11	79
num_il_tl	1251571	8.514671	7.352151	0	3	7	11	159
num_op_rev_tl	1251571	8.319066	4.541131	0	5	7	11	91
num_rev_accts	1251570	14.68883	8.080868	0	9	13	19	118
num_rev_tl_bal_gt_0	1251571	5.731843	3.253593	0	3	5	7	59
num_sats	1263257	11.70843	5.475699	0	8	11	14	97
num_tl_120dpd_2m	1202848	0.000865	0.031176	0	0	0	0	6
num_tl_30dpd	1251571	0.003876	0.0664	0	0	0	0	4
num_tl_90g_dpd_24m	1251571	0.090736	0.506308	0	0	0	0	39
num_tl_op_past_12m	1251571	2.109639	1.804914	0	1	2	3	32
pct_tl_nvr_dlq	1251418	94.0675	8.79095	0	91.1	97.6	100	100
percent_bc_gt_75	1258823	47.45467	35.83138	0	16.7	50	75	100
pub_rec_bankruptcies	1320482	0.126787	0.372602	0	0	0	0	12
tax_liens	1321742	0.056615	0.418396	0	0	0	0	85
tot_hi_cred_lim	1251571	173672.7	176807.8	0	49789	112389	250854.5	9999999
total_bal_ex_mort	1271817	50299.27	47793.73	0	21430	37888	63251	2921551
total_bc_limit	1271817	21487.9	21304.29	0	7800	15000	28000	1105500
total_il_high_credit_limit	1251571	42211.78	43305.29	0	14825	31728	56712	2101913

## SUMMARY OBSERVATIONS POST CLEANING

	count	mean	std	min	25%	50%	75%	max
member_id	0							
loan_amnt	1321847	14748.22	8622.143	500	8000	12900	20000	40000
funded_amnt	1321847	14739.22	8618.488	500	8000	12850	20000	40000
funded_amnt_inv	1321847	14710.71	8625.729	0	8000	12800	20000	40000
installment	1321847	439.1395	253.2311	14.01	255.9	380.25	578.31	1584.9
emp_length	1321847	5.769954	3.726066	0	2	6	10	10
url	0							
zip_code	1321847	51004.38	31154.91	700	22900	47200	80100	99900
delinq_2yrs	1321847	0.329587	0.892034	0	0	0	0	39
inq_last_6mths	1321847	0.65086	0.957657	0	0	0	1	33
mths_since_last_delinq	1321847	16.99237	22.9671	0	0	0	31	195
mths_since_last_record	1321847	11.48035	27.93378	0	0	0	0	129
open_acc	1321847	11.65589	5.469351	0	8	11	14	97
pub_rec	1321847	0.215023	0.617157	0	0	0	0	86
revol_bal	1321847	16928.01	22933.9	0	6295	11652	20584	2904836
total_acc	1321847	25.02163	11.90399	0	16	23	32	176
out_prncp	1321847	5971.41	7531.106	0	0	2927.55	9990.595	40205.27
out_prncp_inv	1321847	5968.963	7528.321	0	0	2926.1	9986.6	40205.27
total_pymnt	1321847	10065.93	8677.782	0	3677.26	7498.63	13686.11	61501.88
total_pymnt_inv	1321847	10038.57	8667.587	0	3660.27	7472.47	13647.27	61414.02
total_rec_prncp	1321847	7790.732	7367.401	0	2408.46	5319.5	10573.48	40000
total_rec_int	1321847	2192.841	2367.291	0	689.14	1430.24	2798.65	26501.88
total_rec_late_fee	1321847	0.629722	5.518013	0	0	0	0	437.99
recoveries	1321847	81.7311	533.0982	0	0	0	0	39444.37
collection_recovery_fee	1321847	12.08557	89.39024	0	0	0	0	7002.19
last_pymnt_amnt	1321847	2840.73	5556.066	0	298.14	505.6	1327.635	42148.53
collections_12_mths_ex_med	1321847	0.016783	0.146714	0	0	0	0	20
policy_code	1321847	1	0	1	1	1	1	1
annual_inc	1321847	76860.81	69223.66	1896	46053.25	65000	91400	9573072
dti	1321847	18.29373	8.379607	-1	12.02	17.8	24.17	69.49
acc_now_delinq	1321847	0.005582	0.081197	0	0	0	0	14
tot_coll_amnt	1321847	228.3459	8216.852	0	0	0	0	9152545
tot_cur_bal	1321847	137653.1	152744.7	0	31586	80847	201466	8000078
total_rev_hi_lim	1321847	32304.04	35754.4	0	14600	24100	39300	9999999
acc_open_past_24mths	1321847	4.530395	3.062295	0	2	4	6	64
avg_cur_bal	1321847	16953.34	21720.21	0	3294	8335	21232	958084
bc_open_to_buy	1321847	14811.81	33012.31	0	1455	4796	13512	9999999
chargeoff_within_12_mths	1321847	0.008962	0.108384	0	0	0	0	10
delinq_amnt	1321847	15.61694	812.8688	0	0	0	0	185408
mo_sin_old_il_acct	1321847	127.308	49.8576	0	104	130	151	724

mo_sin_old_rev_tl_op	1321847	183.4455	92.37072	2	121	167	229	901
mo_sin_rcnt_rev_tl_op	1321847	13.15444	16.39398	0	4	8	15	438
mo_sin_rcnt_tl	1321847	7.975982	8.834012	0	3	6	10	314
mort_acc	1321847	1.663525	1.979451	0	0	1	3	61
mths_since_recent_bc	1321847	24.12318	30.83404	0	6	14	28	639
mths_since_recent_bc_dlq	1321847	9.503957	20.22588	0	0	0	0	195
mths_since_recent_inq	1321847	5.892125	5.98367	0	1	4	9	25
mths_since_recent_revol_delinq	1321847	12.03261	21.27011	0	0	0	17	197
num_accts_ever_120_pd	1321847	0.47844	1.279002	0	0	0	0	51
num_actv_bc_tl	1321847	3.516756	2.356072	0	2	3	5	47
num_actv_rev_tl	1321847	5.470818	3.494211	0	3	5	7	59
num_bc_sats	1321847	4.535068	3.037534	0	3	4	6	71
num_bc_tl	1321847	7.71718	5.005622	0	4	7	10	79
num_il_tl	1321847	8.061988	7.404718	0	3	6	11	159
num_op_rev_tl	1321847	7.876783	4.796799	0	5	7	10	91
num_rev_accts	1321847	13.90789	8.52584	0	8	13	18	118
num_rev_tl_bal_gt_0	1321847	5.42711	3.417147	0	3	5	7	59
num_sats	1321847	11.18946	5.870372	0	7	10	14	97
num_tl_120dpd_2m	1321847	0.000787	0.029741	0	0	0	0	6
num_tl_30dpd	1321847	0.00367	0.064617	0	0	0	0	4
num_tl_90g_dpd_24m	1321847	0.085912	0.493085	0	0	0	0	39
num_tl_op_past_12m	1321847	1.99748	1.818943	0	1	2	3	32
pct_tl_nvr_dlq	1321847	89.05552	22.79273	0	89.5	97	100	100
percent_bc_gt_75	1321847	45.19209	36.39951	0	0	44.4	75	100
pub_rec_bankruptcies	1321847	0.126656	0.372432	0	0	0	0	12
tax_liens	1321847	0.05661	0.41838	0	0	0	0	85
tot_hi_cred_lim	1321847	170414.6	172592.2	0	51914	112389	241673.5	9999999
total_bal_ex_mort	1321847	49829.52	46940.33	0	22044	37888	61817	2921551
total_bc_limit	1321847	21242.34	20933.88	0	8000	15000	27200	1105500
total_il_high_credit_limit	1321847	41654.41	42204	0	15624	31728	54762	2101913
90day_worse_rating	1321847	0.262762	0.440134	0	0	0	1	1

## HANDLING MISSING DATA: LOAN DATA

Missing Data analysis was undertaken in two phases:

1. Handling Numeric columns
2. Handling text columns

Handling Numeric Columns:

### Step 1:

For those columns whose distribution doesn't change much on replacing with zero, the same was done.

**delinq\_2yrs** has 29 missing observations and, we can replace those with zero, giving lenders the benefit of the doubt they wouldn't forget someone delinquent.

**inq\_last\_6mths** was done in a similar manner.

**mths\_since\_last\_delinq** is also replaced by 0 because of the same reason.

Other columns imputed with zeroes are - **open\_acc, pub\_rec, total\_acc, collections\_12\_mths\_ex\_med, acc\_now\_delinq**

### Step 2:

Numeric columns where missing values can be replaced by median without changing the distribution of the data, we can replace it by median.

Example:

**annual\_inc** has only 4 missing observations so we do a median value imputation with this feature.

**tot\_coll\_amt** will involve a median value imputation.

```
fullData['tot_coll_amt'] = fullData['tot_coll_amt'].fillna(fullData['tot_coll_amt'].median())
```

Other columns for median replacement:

### Step 3:

We also included a few derived columns to get some necessary insights and information:

**mths\_since\_last\_major\_derog** will be changed to a new variable where missing values = 0

for no derogs and non-missing = 1 for atleast 1 derog. feature will be named

**90day\_worse\_rating**

```
fullData['90day_worse_rating'] = np.where(fullData['mths_since_last_major_derog'].isnull(), 0,
```

1) Joint Account Type and Individual Account Type were mutually exclusive. So were the incomes. Hence they were put together to form a single column

open to buy = credit limit - (sum of holds and outstanding balance) assuming that sum of holds and outstanding balance is zero in this case

```
fullData['bc_open_to_buy'].fillna(fullData['tot_hi_cred_lim'], inplace=True)
```

With the high & low credit ranges, we were able to calculate the risk score

```
fullData['risk_score'] = fullData[['fico_range_low', 'fico_range_high']].mean(axis=1)
```

**Step 4:** Delete those insignificant columns where most of the column is empty and will not let us analyze anything

features below are being dropped due to their significantly high proportion of missing values or they are date values.

```
fullData = fullData.drop(['earliest_cr_line', 'last_pymnt_d', 'next_pymnt_d',
'last_credit_pull_d',
'annual_inc_joint', 'dti_joint', 'verification_status_joint', 'open_acc_6m', 'open_il_6m',
'open_il_12m', 'open_il_24m', 'mths_since_rcnt_il',
'total_bal_il', 'il_util', 'open_rv_24m', 'open_rv_12m', 'max_bal_bc',
'all_util', 'total_cu_tl',
'mths_since_last_record', 'mths_since_last_major_derog'])
```

Handling Missing & Cleaning for Text Columns:

**STEP 1:** Check all the text columns that are categorical. We noticed that there weren't any missing values

-> Split Issue Date to obtain Month and Year columns.

-> Home ownership had about 7 categories. Merged 'other', 'none' and 'any' into a single category called none.

-> Employment length was changed to contain numeric values

-> The same was done with term and interest rate columns

**STEP 2:** A lot of columns seemed like they couldn't give any important information.

These were dropped - url, dech, emp title

## CLEANING: DECLINED LOAN DATA SET

After downloading the Declined Loan Data Set, we observed that there were 9 columns:

Amount  
Requested  
Application Date  
Loan Title  
Risk Score  
Debt-To-Income  
Ratio  
Zip Code  
State  
Employment  
Length  
Policy Code

- 1) **Loan Title:** We removed the Loan Title Column since we found a lot of titles that didn't make any sense and realized that it wouldn't help with any of our further analysis
- 2) **Application Date:** We split the application date into Year, Month and Day since we plan on using these columns for future Analysis
- 3) **State:** We replaced all the missing State values with 'NA'
- 4) **Zip Code:** We replaced the the XX's in Zip Code with 00's since there's no way of knowing the last 2 digits. We plan on removing the column in future since it doesn't make much sense
- 5) **Risk Score:** We replaced the missing values with '999' since more than 50% of the data was missing and replacing the values with either a mean or median would change the distribution by a large margin



SUMMARY OBSERVATIONS MADE AFTER REPLACING THE MISSING VALUES OF THE RISK SCORE WITH MEDIAN AND MEAN. REPLACED THE MISSING VALUES WITH 999

	Amount Requested	Risk_Score	Employment Length	Policy Code
count	11079386.000000	4676607.000000	11079386.000000	11079386.000000
mean	13391.543411	623.382936	1.707245	0.005543
std	16196.710716	108.408128	1.884364	0.105140
min	0.000000	0.000000	0.000000	0.000000
25%	4500.000000	591.000000	1.000000	0.000000
50%	10000.000000	640.000000	1.000000	0.000000
75%	20000.000000	678.000000	1.000000	0.000000
max	1400000.000000	990.000000	10.000000	2.000000

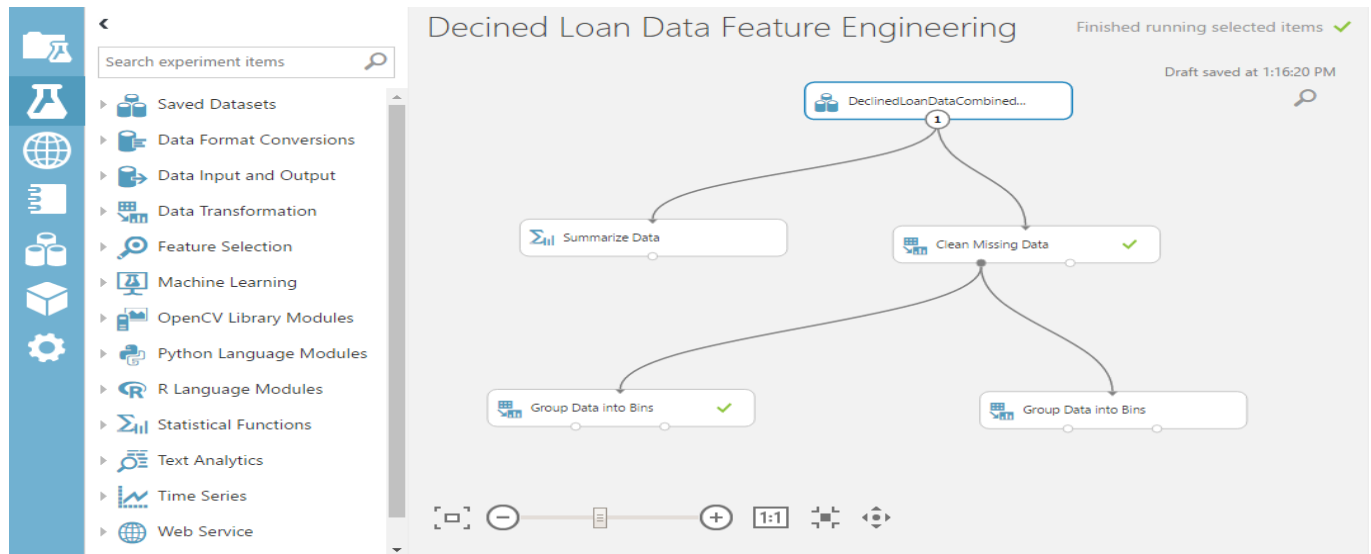
	Amount Requested	Risk_Score	Employment Length	Policy Code
count	11079386.000000	11079386.000000	11079386.000000	11079386.000000
mean	13391.543411	632.985940	1.707245	0.005543
std	16196.710716	70.908453	1.884364	0.105140
min	0.000000	0.000000	0.000000	0.000000
25%	4500.000000	640.000000	1.000000	0.000000
50%	10000.000000	640.000000	1.000000	0.000000
75%	20000.000000	640.000000	1.000000	0.000000
max	1400000.000000	990.000000	10.000000	2.000000

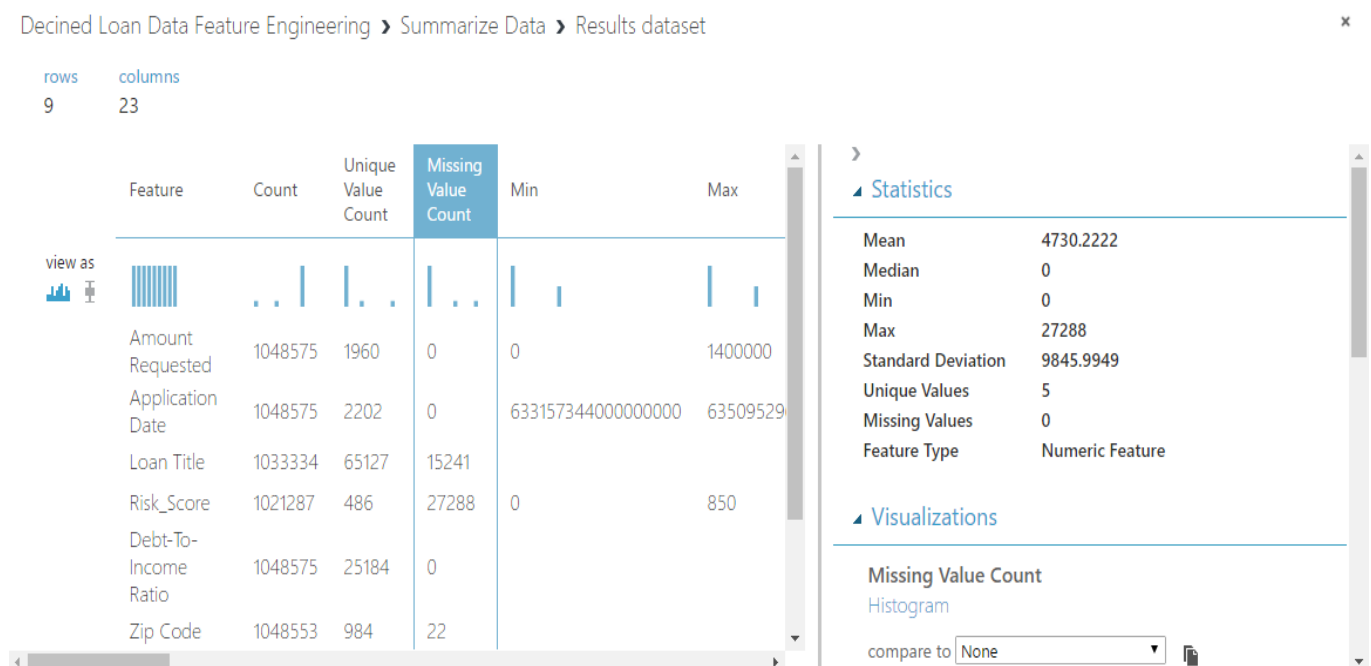
	Amount Requested	Risk_Score	Employment Length	Policy Code
count	11079386.000000	11079386.000000	11079386.000000	11079386.000000
mean	13391.543411	632.985940	1.707245	0.005543
std	16196.710716	70.908453	1.884364	0.105140
min	0.000000	0.000000	0.000000	0.000000
25%	4500.000000	640.000000	1.000000	0.000000
50%	10000.000000	640.000000	1.000000	0.000000
75%	20000.000000	640.000000	1.000000	0.000000
max	1400000.000000	990.000000	10.000000	2.000000



## MICROSOFT AZURE ML STUDIO TO CROSS VALIDATE OUR FINDINGS AND VISUALIZE CLEANING, SUMMARY, AND FEATURE ENGINEERING



## CHECKING FOR MISSING VALUES USING SUMMARIZE DATA USING MICROSOFT AZURE ML STUDIO

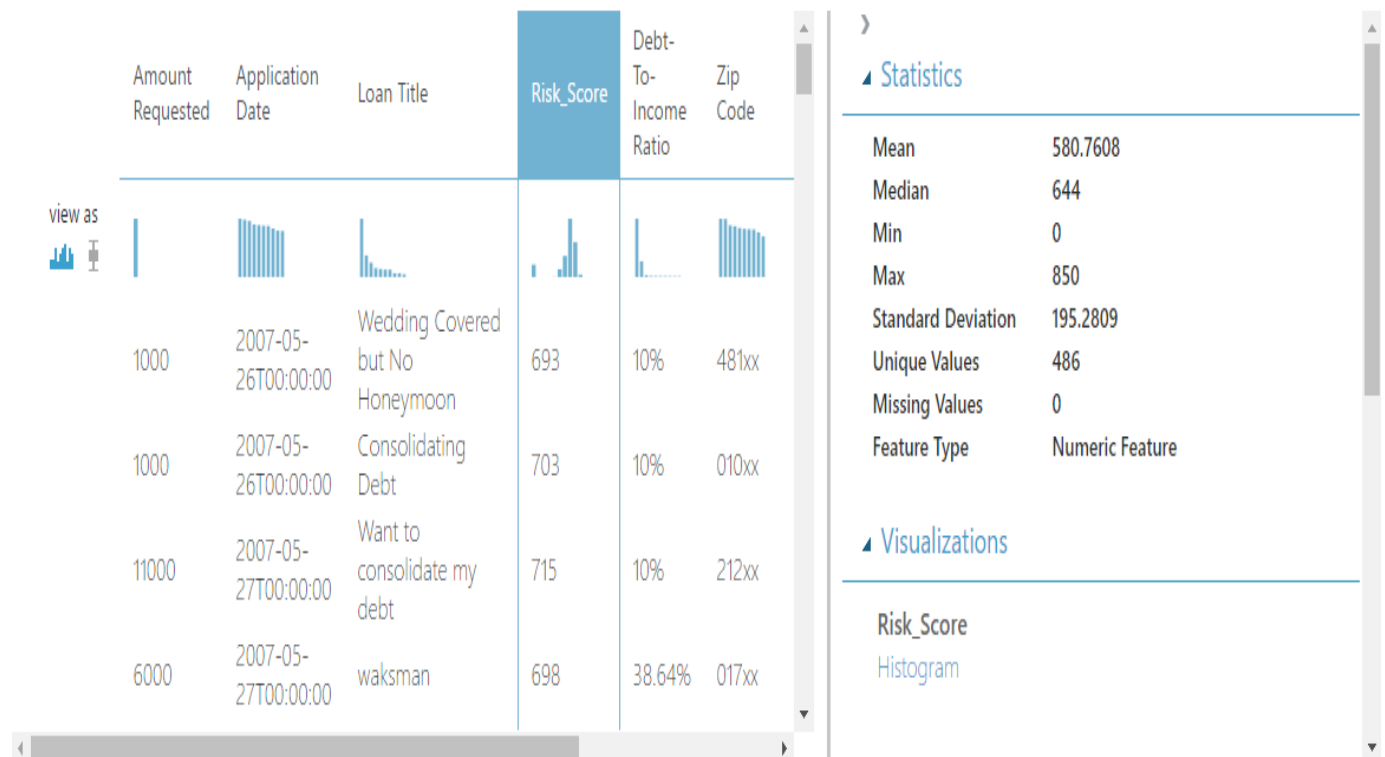


## CLEANING DATA AND CHECKING FOR MISSING VALUES USING MICROSOFT AZURE ML STUDIO

Decided Loan Data Feature Engineering > Clean Missing Data > Cleaned dataset

rows  
1048575

columns  
9

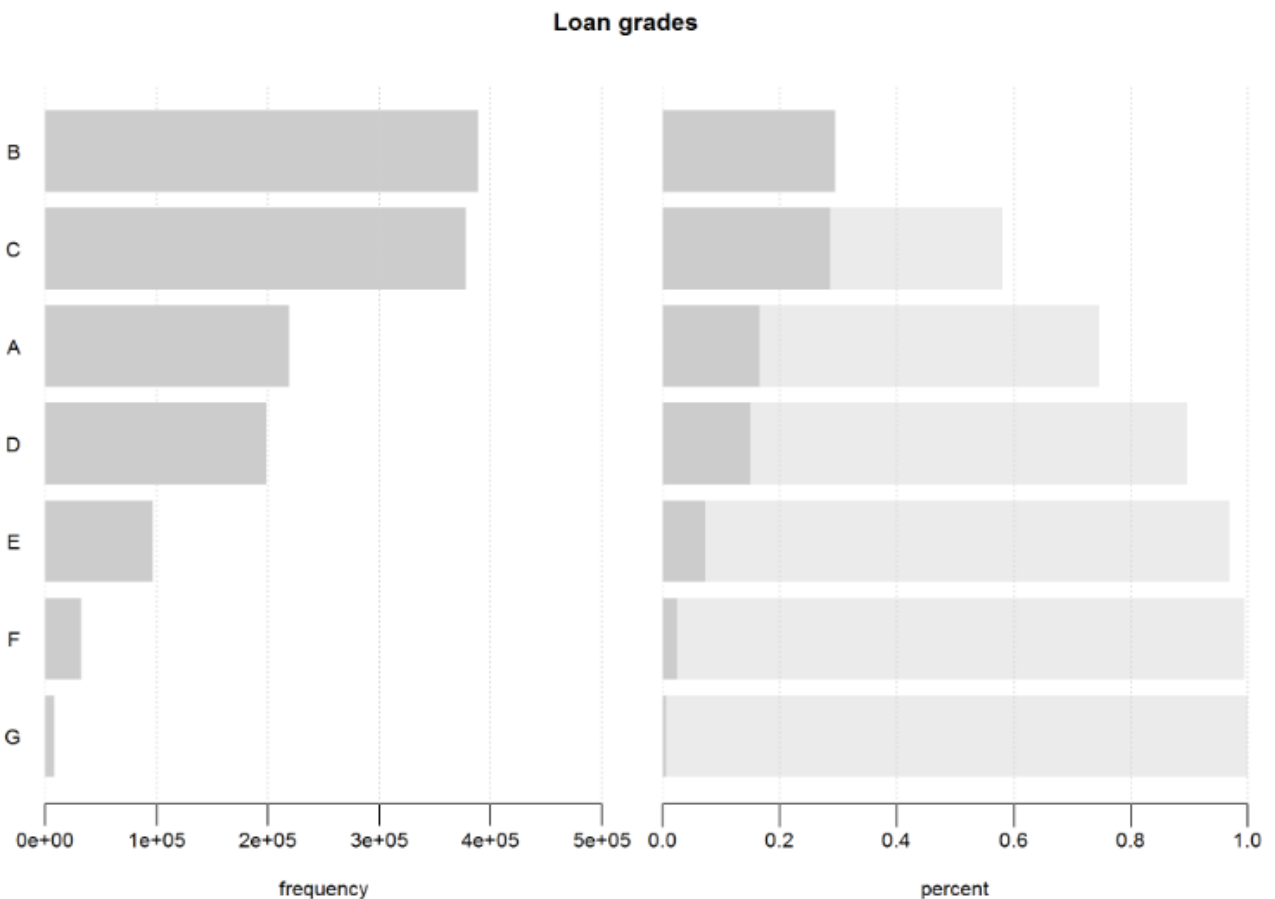


## EXPLORATORY DATA ANALYSIS: LOAN DATA

USING JUPYTER NOTEBOOK AND R **PUBLISHED LINK:** [HTTP://RPUBS.COM/PALECANDA](http://rpubs.com/palecanda)

## ANALYSIS FOR LOAN DATA: GRADES

LOAN DATA SET GRADE FREQUENCY



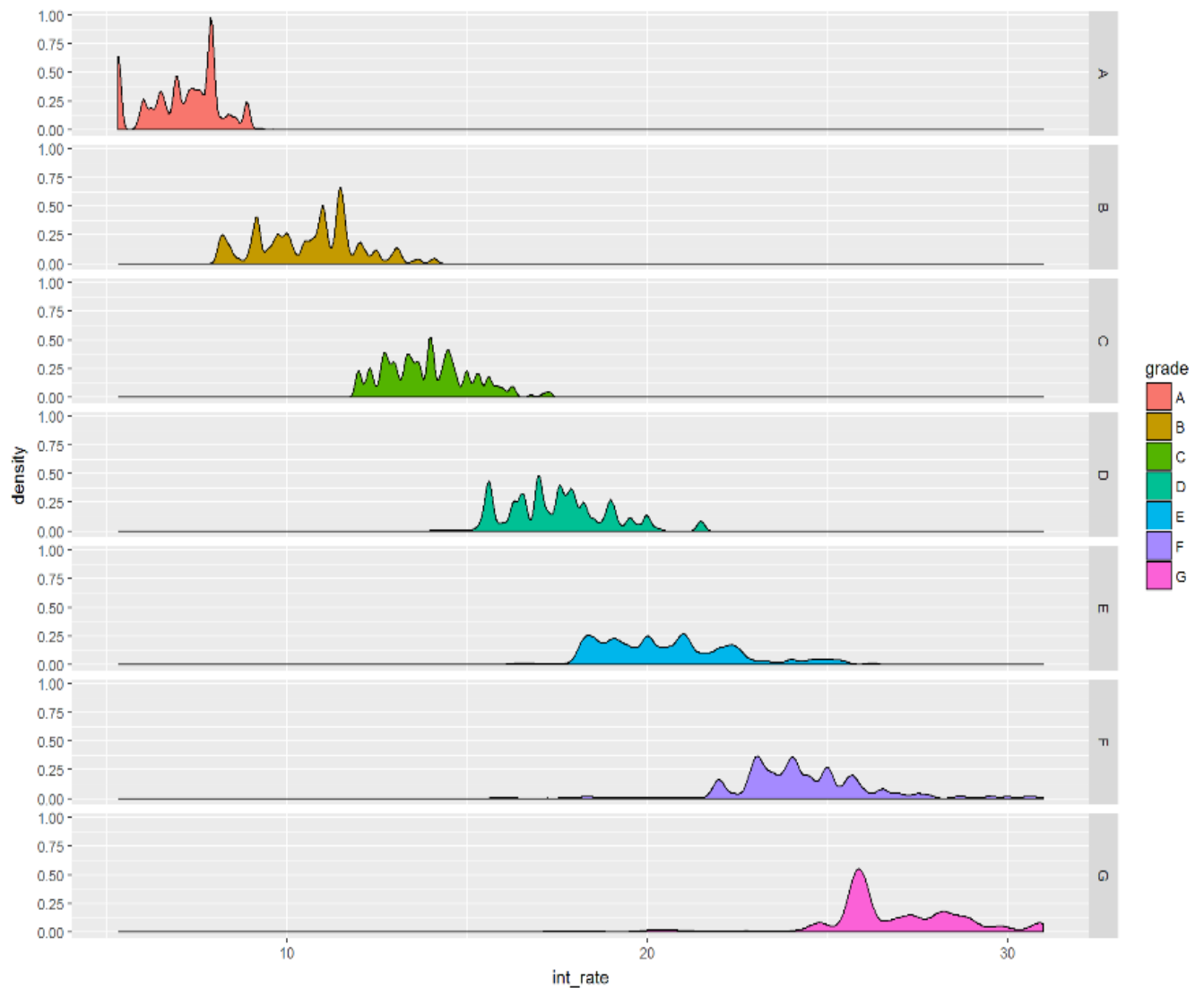
```
library(plotly)
library(ggplot2)
library(plyr)
install.packages("DescTools")
library(DescTools)

#Loan Status Frequency
Desc(data1$loan_status, plotit = T)

#Loan Grade Frequency
Desc(data1$grade, main = "Loan grades", plotit = TRUE)
```

By checking the frequency of the grade, we see that people with lower grades (F/G) are very less as compared to people with Higher grades (A,B,C). Hence, people with a good credit score are more in number

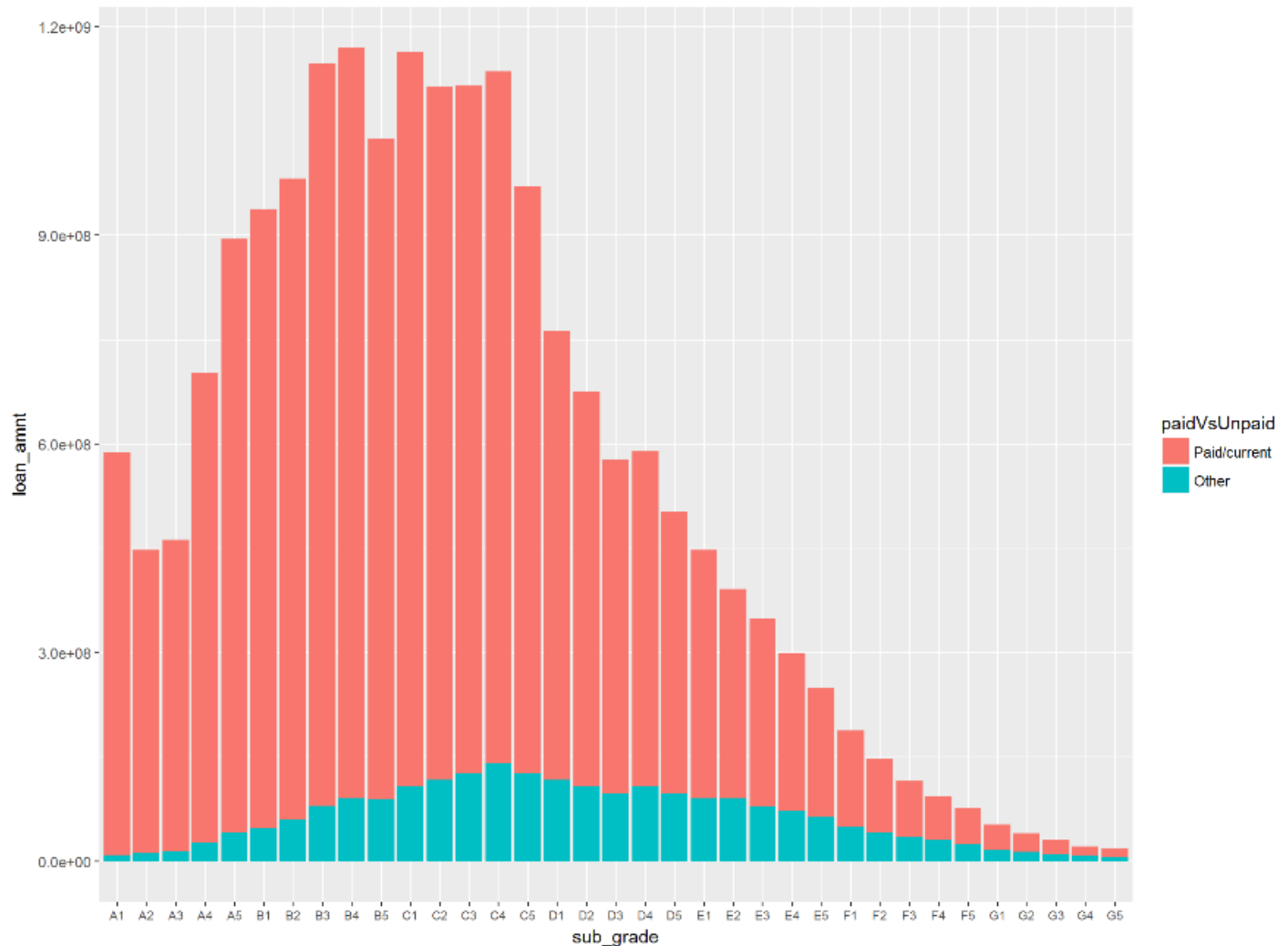
#### EXPLORING INTEREST RATES BASED ON GRADES ASSIGNED BY LENDING CLUB



```
library(ggplot2)
ggplot(data1, aes(int_rate, fill = grade)) + geom_density() + facet_grid(grade ~ .)
```

Grades are assigned based on risk, and so interest rates go up as the risk goes up. Here we see that the interest rate is the highest for grades F and G, since their credit score, hence risk score is low

## PAID VS. UNPAID LOAN AMOUNT OVER THE GRADES

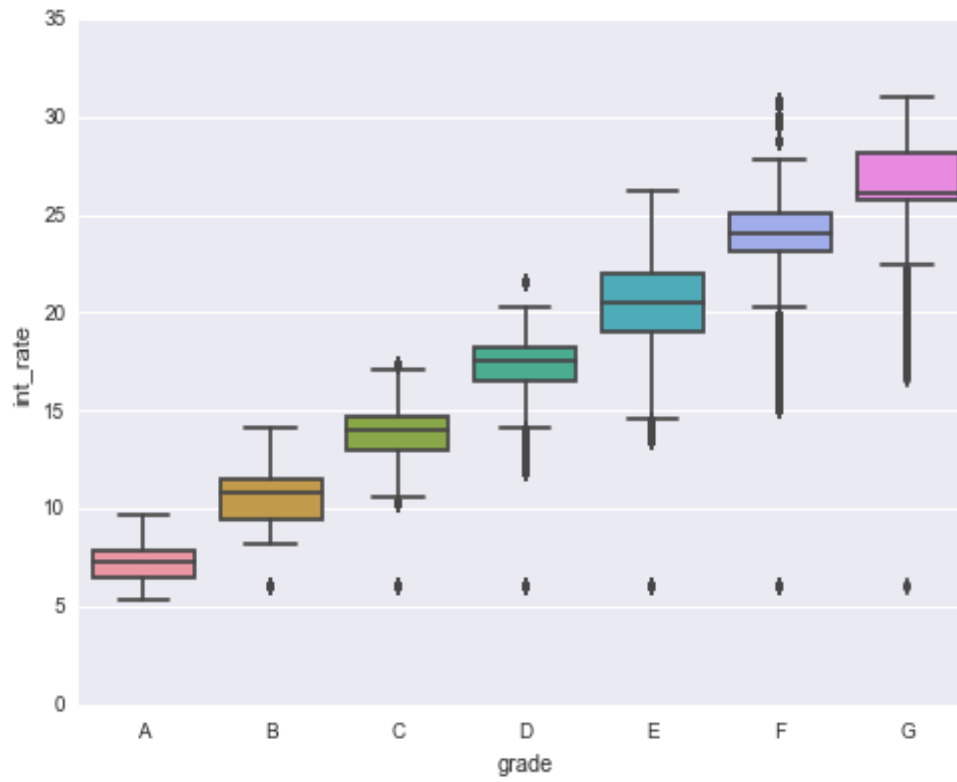


```
data1$paidvsunpaid <- "other"
data1$paidvsunpaid[which(data1$loan_status == "Fully Paid" | data1$loan_status == "current" | data1$loan_status ==
                          "Does not meet the credit policy. Status:Fully Paid")] <- "Paid/current"
data1$paidvsunpaid <- factor(data1$paidvsunpaid)
data1$paidvsunpaid <- factor(data1$paidvsunpaid, levels = rev(levels(data1$paidvsunpaid)))
table(data1$paidvsunpaid)
ggplot(data1, aes(paidvsunpaid, loan_amnt, fill = paidvsunpaid)) + geom_bar(stat = "identity")

#Paid vs. Unpaid loan amount over the Grades
loan_by_grade <- aggregate(loan_amnt ~ sub_grade + paidvsunpaid, data = data1, sum)
gbar <- ggplot(loan_by_grade, aes(sub_grade, loan_amnt, fill = paidvsunpaid))
gbar + geom_bar(stat = "identity") + theme(axis.text.x=element_text(size=7))
```

Here, we have created new column with 2 factor levels. 1) "Paid/current" - Represents the status is Current or Fully Paid. 2) "Other" - Represents defaults, charger-off and other status

## INTEREST RATES AGAINST GRADES



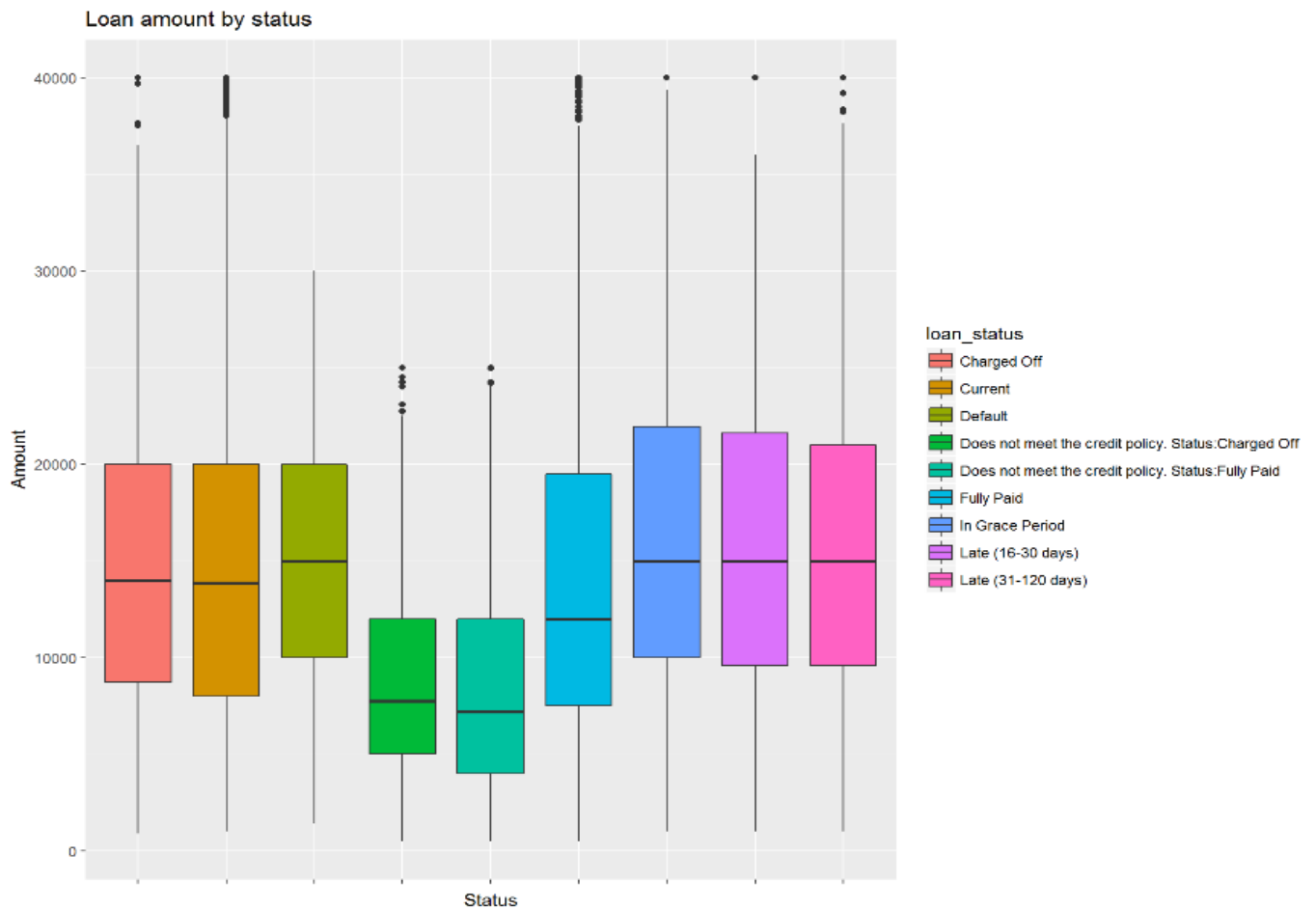
```
|
import seaborn as sns
import matplotlib.pyplot as plt

ax1 = sns.boxplot(x='grade', y='int_rate', data=loan,
                  order=sorted(loan['grade'].unique()))
ax1.set_ybound(lower=0)
plt.show()
```

**We observe that the loans have a higher rate of interest until one gets to the higher grades such as A and B**

## ANALYSIS FOR LOAN DATA: LOAN AMOUNT

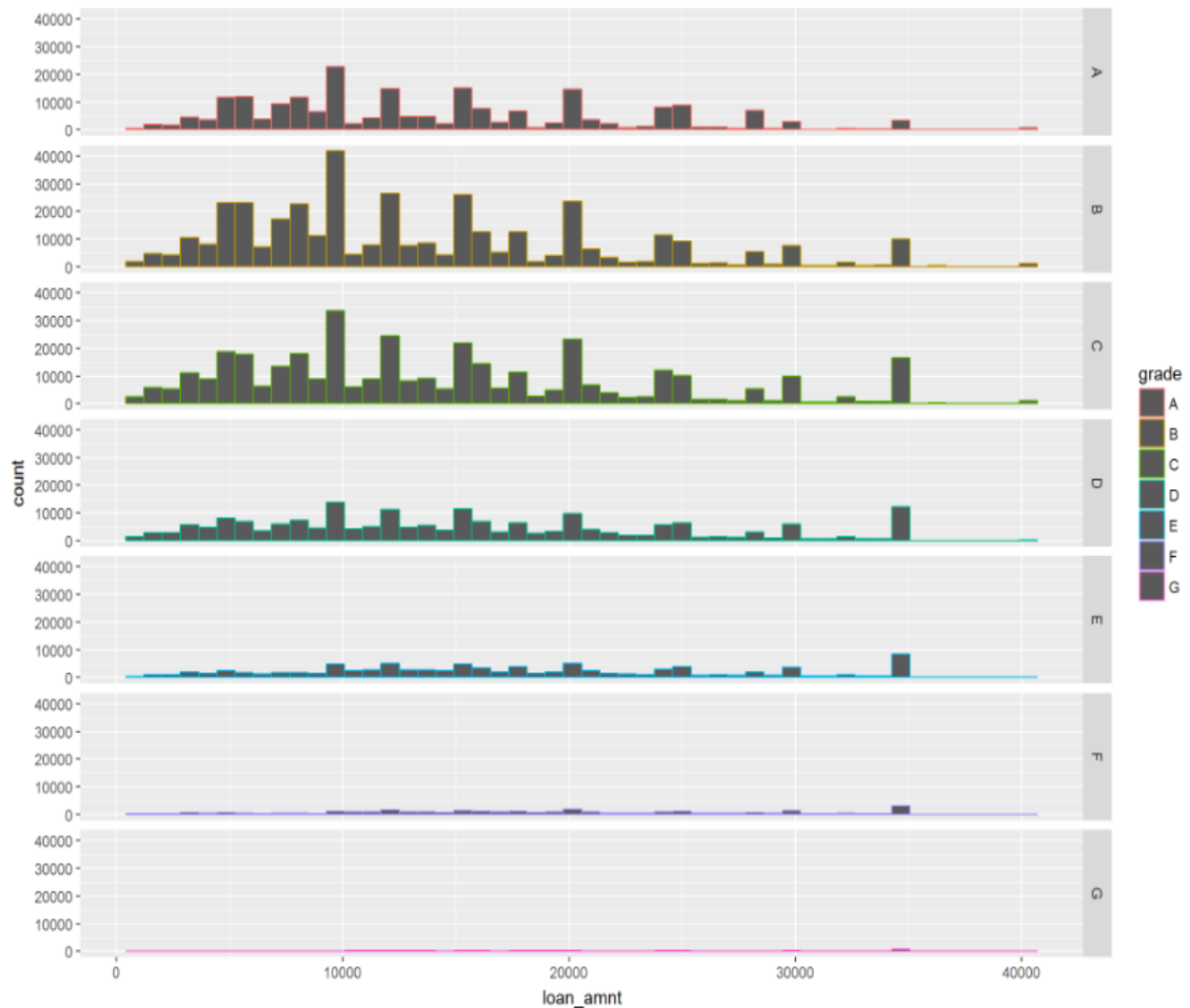
### LOAN AMOUNT AGAINST LOAN STATUS



```
#checking the distribution of loan amounts by status.
library(ggplot2)
box_status <- ggplot(data1, aes(loan_status, loan_amnt))
box_status + geom_boxplot(aes(fill = loan_status)) +
  theme(axis.text.x = element_blank()) +
  labs(list(
    title = "Loan amount by status",
    x = "Status",
    y = "Amount"))
```

By plotting the loan amount against the loan status, we see that most of the loans are in grace period, or are late

## COUNT OF LOAN AMOUNT AGAINST GRADE



```
library(ggplot2)
library(dplyr)
library(plotly)
library(ggplot2)
```

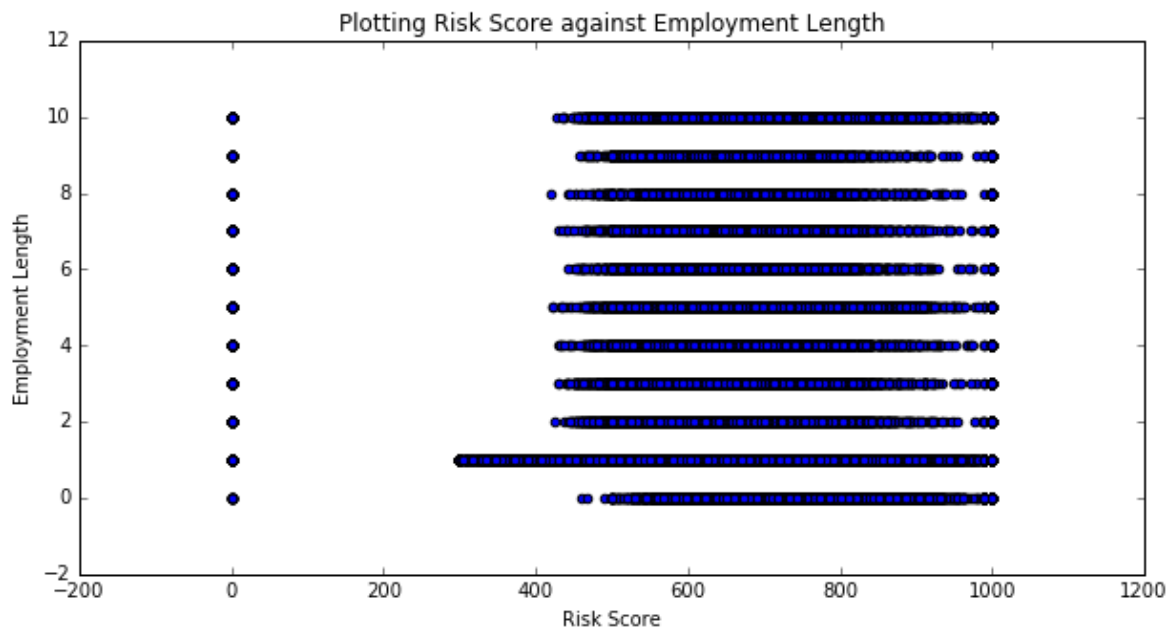
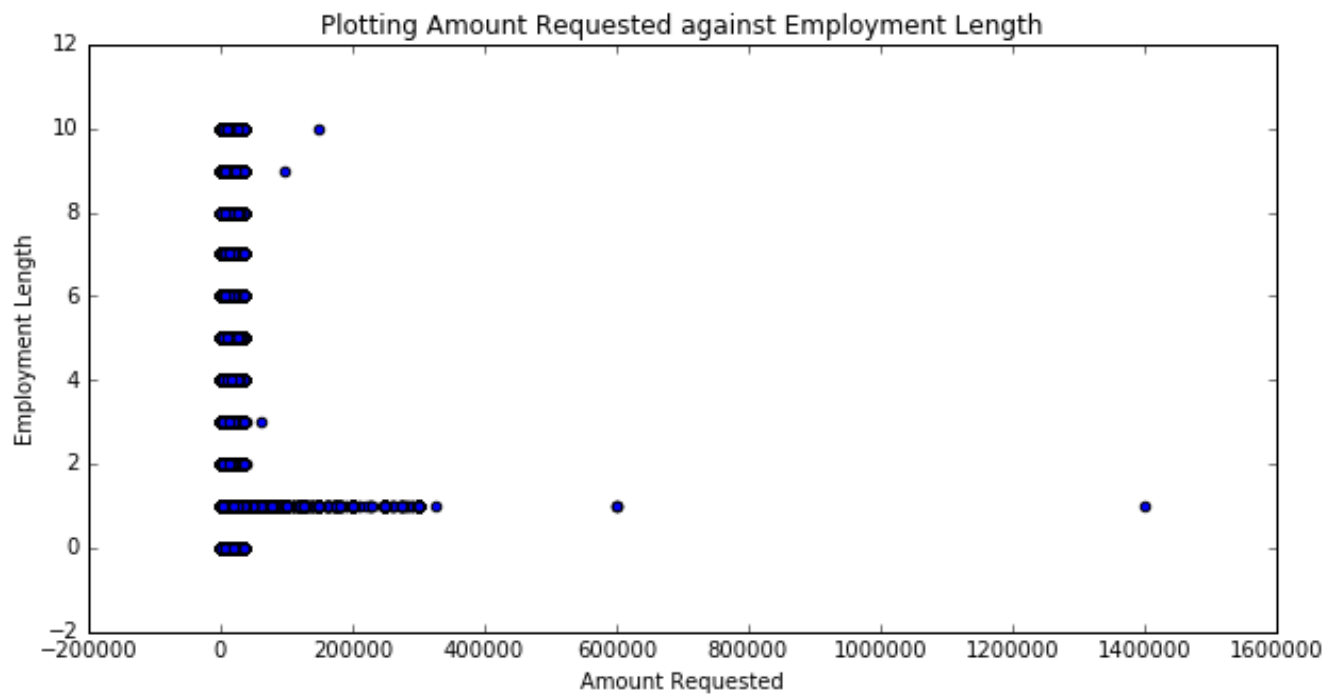
```
#Loan amount Distribution based on Grades assigned by Lending Club
#Those with higher grades (A, B, C and D) have received more loans compared to those with lower grades (E, F and G)
library(ggplot2)
ggplot(data1, aes(loan_amnt, col = grade)) + geom_histogram(bins = 50) + facet_grid(grade ~ .)
```

**We observe that, those with higher grades (A, B, C and D) have received more loans compared to those with lower grades (E, F and G)**



## EXPLORATORY ANALYSIS: DECLINED LOAN DATA

## REJECTED LOAN AMOUNT AGAINST EMPLOYMENT LENGTH VS RISK SCORE



In this Analysis for Declined Loan Data, we have analyzed Employment length against the Amount Requested and Employment length against the Risk Score. We can see that the Loan Requested has mostly been declined for people with an Employment Length between 0-2 years. This may be due to various reasons

- 1) Their credit scores are low since they just started working
- 2) After plotting the employment length against the risk score, the above point is pretty evident since the risk score is calculated using the credit scores (FICO scores/2)

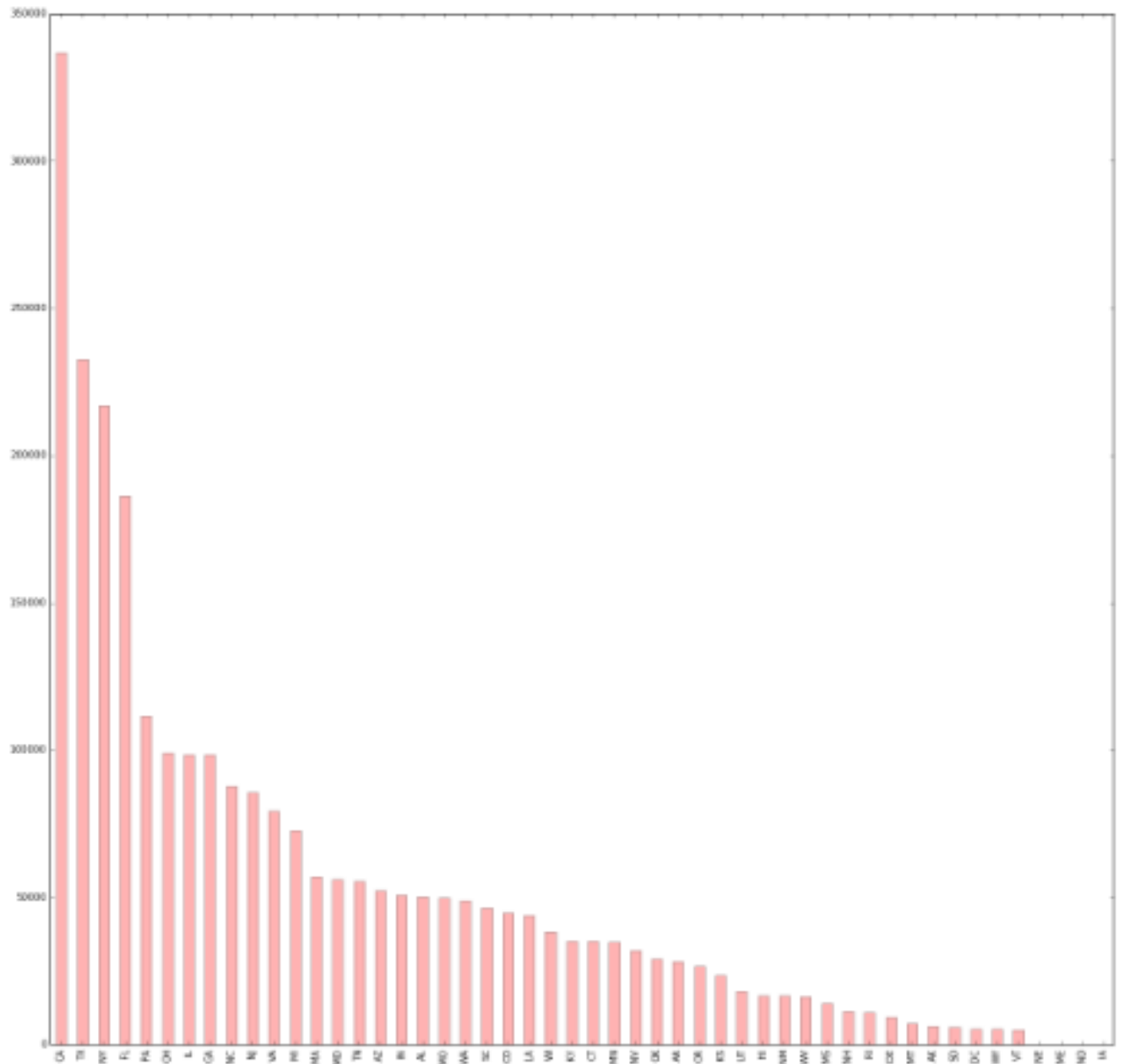
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

#Plotting the Amount Requested against Employment Length
%matplotlib inline
plt.figure(figsize=(10,5))
plt.scatter(fullData['Amount Requested'], fullData['Employment Length'])
plt.title("Amount Requested against Employment Length")
plt.ylabel('Employment Length')
plt.xlabel('Amount Requested')
plt.show()
```

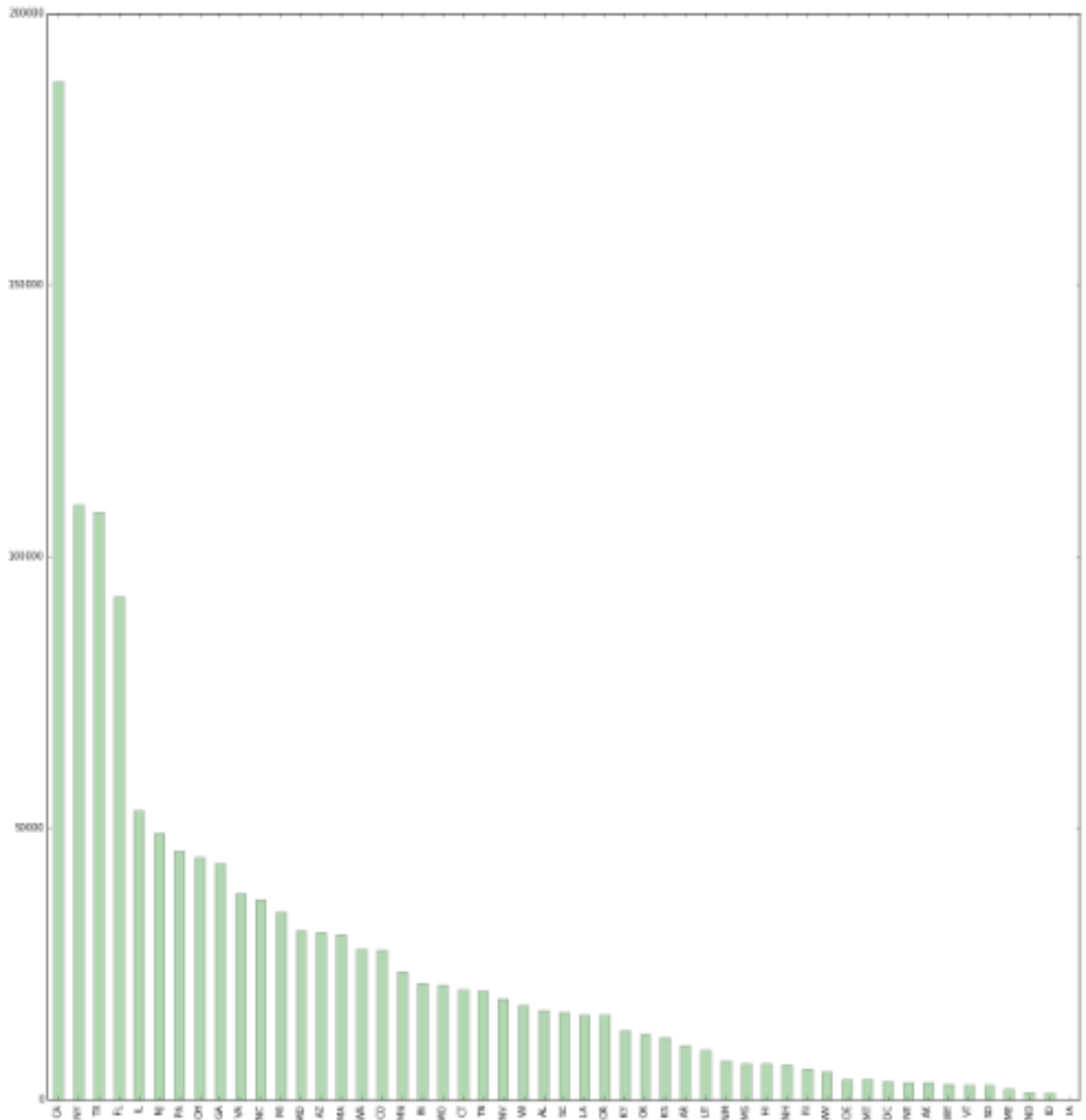
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

#Plotting Risk Score against Employment Length
%matplotlib inline
plt.figure(figsize=(10,5))
plt.scatter(fullData['Risk_Score'], fullData['Employment Length'])
plt.title("Plotting Risk Score against Employment Length")
plt.ylabel('Employment Length')
plt.xlabel('Risk Score')
plt.show()
```

## COUNT OF ACCEPTED AND REJECTED LOAN AMOUNT AGAINST STATES



```
fig = plt.figure(figsize=(20, 20), dpi=100)
fullData['State'].value_counts().plot(kind='bar', alpha=.30, color='red')
```



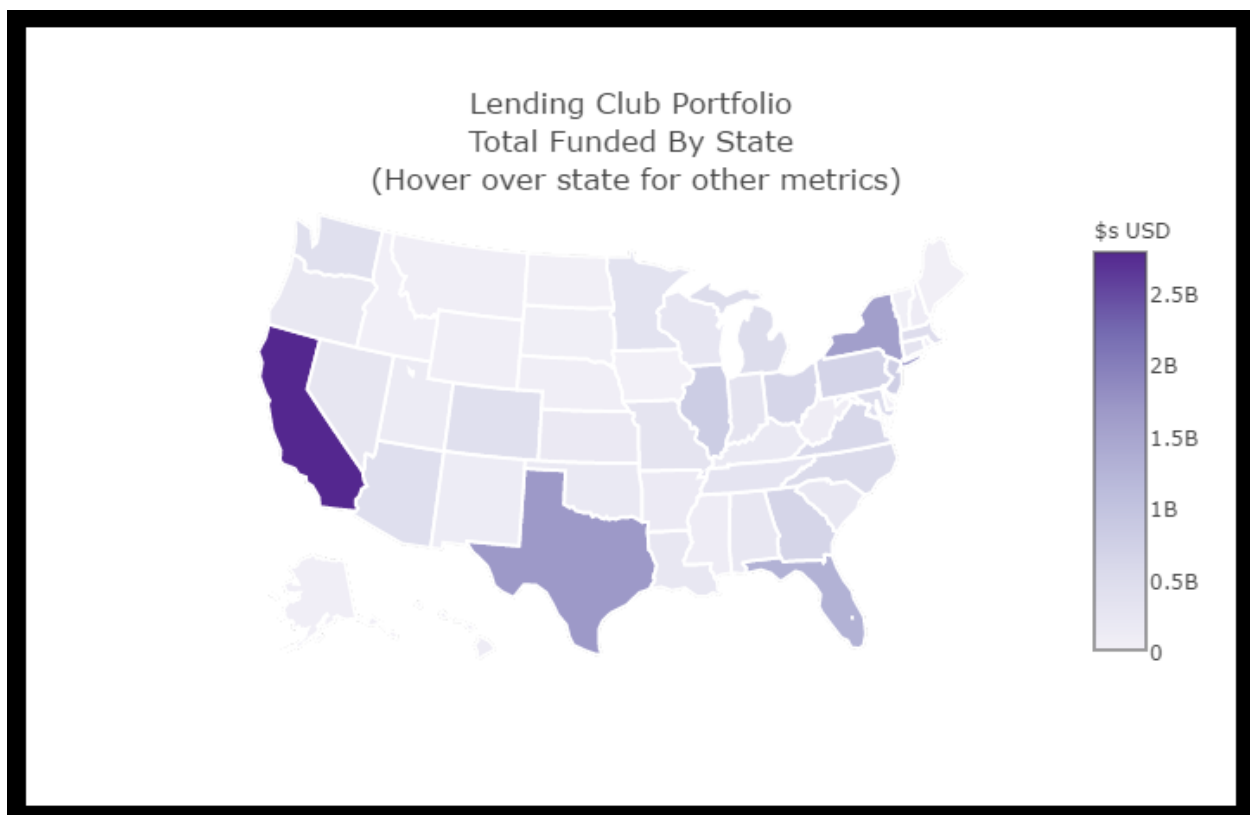
The above 2 graphs shows that the count of the accepted and rejected loan are high in CA, TX, NY and FL. T the reason for this could be that these states were the first to be eligible to get a loan from lending club.

```
import pandas as pd
import numpy as np
from datetime import datetime
from matplotlib import pyplot as plt

%matplotlib inline
loandata = pd.read_csv("C:/Users/taj/Desktop/cleaned_loandata.csv", encoding = "ISO-8859-1", low_memory=False)

fig = plt.figure(figsize=(20, 20), dpi=100)
loandata['addr_state'].value_counts().plot(kind='bar', alpha=.30, color='green')
```

## TOTALS FUNDED BY STATE



The above graph gives a detailed analysis of the Average Balance per Borrower and Average Income per borrower in each state. We see that the Averages are the highest in CA

```

import plotly.plotly as py
import plotly.graph_objs as graph_objs

for col in df_plot.columns:
    df_plot[col] = df_plot[col].astype(str)

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

df_plot['text'] = df_plot['code'] + '<br>' +\
    'Avg Balance Per Borrower ($ USD): '+df_plot['Average_Balance']+'<br>'+\
    'Avg Annual Income Per Borrower ($ USD): '+df_plot['Average_Income']

data = [ dict(
    type='choropleth',
    colorscale = scl,
    autocolorscale = False,
    locations = df_plot['code'],
    z = df_plot['Loan_Funding'],
    locationmode = 'USA-states',
    text = df_plot['text'],
    marker = dict(
        line = dict (
            color = 'rgb(255,255,255)',
            width = 2
        ) ),
    colorbar = dict(
        title = "$s USD"
    ) ]

layout = dict(
    title = 'Lending Club Portfolio<br> Total Funded By State <br> (Hover over state for other metrics)',
    geo = dict(
        scope='usa',
        projection=dict( type='albers usa' ),
        showlakes = True,
        lakecolor = 'rgb(255, 255, 255)',
    )

fig = dict( data=data, layout=layout )
iplot( fig, filename='d3-choropleth-map' )

```

## FEATURE ENGINEERING: PART 1

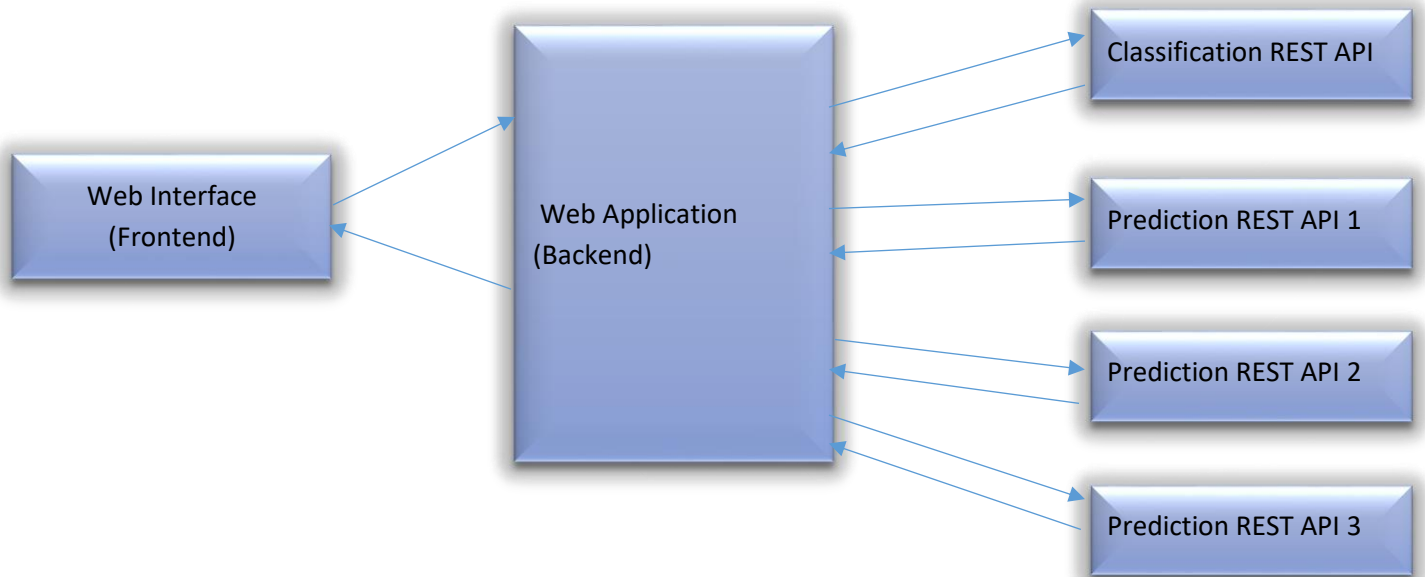
We plan on building models and predicting the best features that we can use to calculate the interest rates.

As of now, we plan on selecting the following features:

- 1) Minimum Credit Score (FICO) – To calculate the Risk Score
- 2) Maximum Credit Score (FICO) – To calculate the Risk Score
- 3) Employment Length – Since we've seen that the employment length affects the credit score
- 4) DTI (Debt-to-Income) – Since we have this in both Loan and Declined Loan Data Set and since the DTI is a vital deciding factor
- 5) State – Since we have this information in both the data sets, and since we've seen that the interest rates, income, etc varies with state (May or May not be included)

# PART 2



**DEPLOYMENT**

1. **Web Interface (HTML, Javascript, JQuery, CSS, Bootstrap)**
  - i. Takes the input and shows the result that if we can provide loan or not (if yes, then at what interest rate).
  - ii. Link to PowerBi Visualizations, Jupyter notebook visualization
  - iii. Link to the results of interest rates from all three APIs as professor said in class.
  - iv. Link to PPT and Report
2. **Web Application (Backend: Java) –**  
Application (Using Java to interacts with the frontend and the 4 rest apis (1 Classification, 3 Prediction using Python and R). Hosting on AWS
3. **REST API on Azure ML Studio**

## CLASSIFICATION

- Created a “**Issue\_Loan\_Flag**” column on the Loan Data and the Rejected Loan Data.  
Value = **0** for declined loan data, **1** for accepted loan data (*except the rows which are marked as policy changed, chargedoff and default - we put 0 for those rows*)
- We ran Logistic Regression, Random Forest, Neural Network and SVM. We selected different percentage of split data for reject loan as the number of rows flagged as reject loan was higher than the number of rows flagged as accepted loan.  
We used 4 features (Loan Amount, Risk score (we assume it is Fico Score), Employment Length and Debt-To-Income ratio.

Model	Split Rejec	Split Lo	True +	False +	True -	False -	Total	Accuracy	True+ %	False + %	True - %	False - %
Logistic Regression	0.1	0.8	152114	607745	9386697	89723	10236279	0.93	1.49	5.94	91.70	0.88
Logistic Regression	0.2	0.8	129315	423439	8460394	112422	9125570	0.94	1.42	4.64	92.71	1.23
Logistic Regression	0.5	0.8	106386	171171	5390214	134310	5802081	0.94	1.83	2.95	92.90	2.31
Logistic Regression	0.8	0.8	96947	60443	2179740	144052	2481182	0.91	3.91	2.44	87.85	5.81
Random Forest	0.1	0.8	224931	274331	9720506	16472	10236240	0.971	2.20	2.68	94.96	0.16
Random Forest	0.2	0.8	219884	205166	8681740	22145	9128935	0.975	2.41	2.25	95.10	0.24
Random Forest	0.5	0.8	207448	102937	5461963	33735	5806083	0.976	3.57	1.77	94.07	0.58
Random Forest	0.8	0.8	199203	45344	2192301	43019	2479867	0.96	8.03	1.83	88.40	1.73
Neural Network	0.1	0.8	0	0	9994669	241462	10236131	0.97	0.00	0.00	97.64	2.36
Neural Network	0.2	0.8	0	0	8886712	241268	9127980	0.97	0.00	0.00	97.36	2.64
Neural Network	0.5	0.8	117441	222605	5339121	123293	5802460	0.94	2.02	3.84	92.01	2.12
Neural Network	0.8	0.8	0	0	2239070	241327	2480397	0.9	0.00	0.00	90.27	9.73
SVM	0.1	0.8	241434	9938917	55445	0	10235796	0.02	2.36	97.10	0.54	0.00
SVM	0.2	0.8	241434	8835224	51201	0	9127859	0.03	2.65	96.79	0.56	0.00
SVM	0.5	0.8	241434	5531644	30971	0	5804049	0.04	4.16	95.31	0.53	0.00
SVM	0.8	0.8	241434	2226369	12439	0	2480242	0.09	9.73	89.76	0.50	0.00

We selected Random Forest (with Split ratio = .5 for reject loan data) as our best model because

- It has Accuracy of .976
  - Lowest False Positive Rate (1.77 %)
  - Higher True Negative Rate (94.07%)
- We created Azure ML web service with the selected model and interacted with it using Java – Spring Web Application.

## ROC CURVES

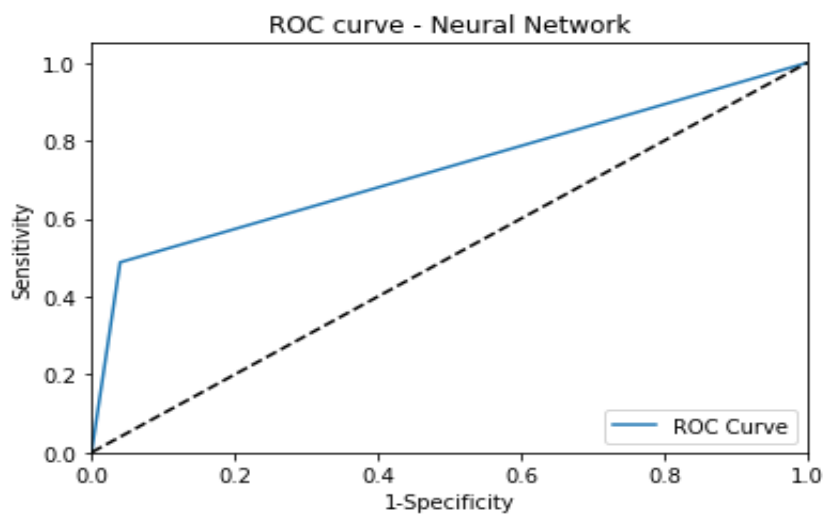
- Neural Network Classification Link:  
[https://nbviewer.jupyter.org/github/pptaj/Assignment2\\_Final/blob/master/Classification/NeuralNetClassification.ipynb](https://nbviewer.jupyter.org/github/pptaj/Assignment2_Final/blob/master/Classification/NeuralNetClassification.ipynb)
- SVM Classification Link:  
[https://nbviewer.jupyter.org/github/pptaj/Assignment2\\_Final/blob/master/Classification/SVMClassification.ipynb](https://nbviewer.jupyter.org/github/pptaj/Assignment2_Final/blob/master/Classification/SVMClassification.ipynb)
- Logistic Regression Link:  
[https://github.com/pptaj/Assignment2\\_Final/blob/master/Classification/LogisticRegression.ipynb](https://github.com/pptaj/Assignment2_Final/blob/master/Classification/LogisticRegression.ipynb)
- Random Forest Link:  
[https://github.com/pptaj/Assignment2\\_Final/blob/master/Classification/RandomForest.ipynb](https://github.com/pptaj/Assignment2_Final/blob/master/Classification/RandomForest.ipynb)

### START OF NEURAL NETWORK CLASSIFIER

```

R-squared error : 0.0923023529502
MEAN ABSOLUTE ERROR : 0.0596123023683
MEAN SQUARED ERROR : 0.0596123023683
MEDIAN ABSOLUTE ERROR : 0.0
0.940387697632
Pred          0          1
Actual
0           5339121    222605
1           123293    117441

```



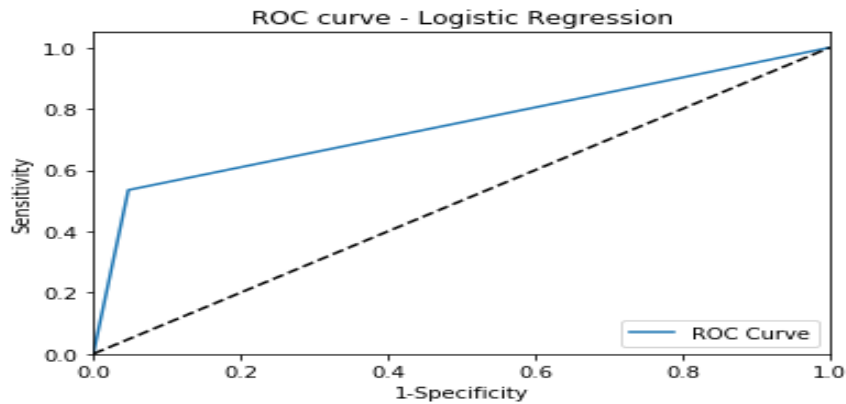
### END OF Neural Network Classifier

---

 START OF LOGISTIC REGRESSION
 

---

R-squared error : 0.122648265343  
 MEAN ABSOLUTE ERROR : 0.0587208251101  
 MEAN SQUARED ERROR : 0.0587208251101  
 MEDIAN ABSOLUTE ERROR : 0.0  
 0.94127917489  
 Pred            0            1  
 Actual  
 0            8460394    423439  
 1            112422    129315




---

 END OF LOGISTIC REGRESSION
 

---

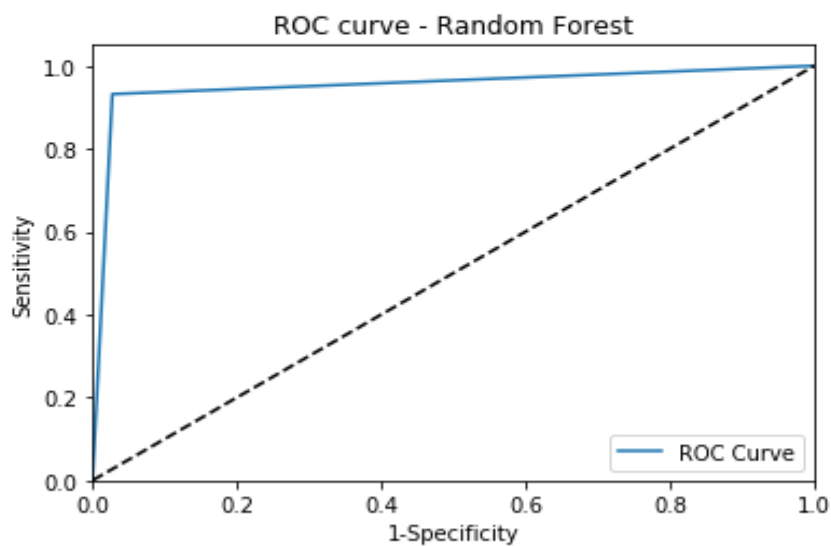


---

 START OF RANDOM FOREST CLASSIFIER
 

---

0.971590838042  
 Pred            0            1  
 Actual  
 0            9720506    274331  
 1            16472    224931



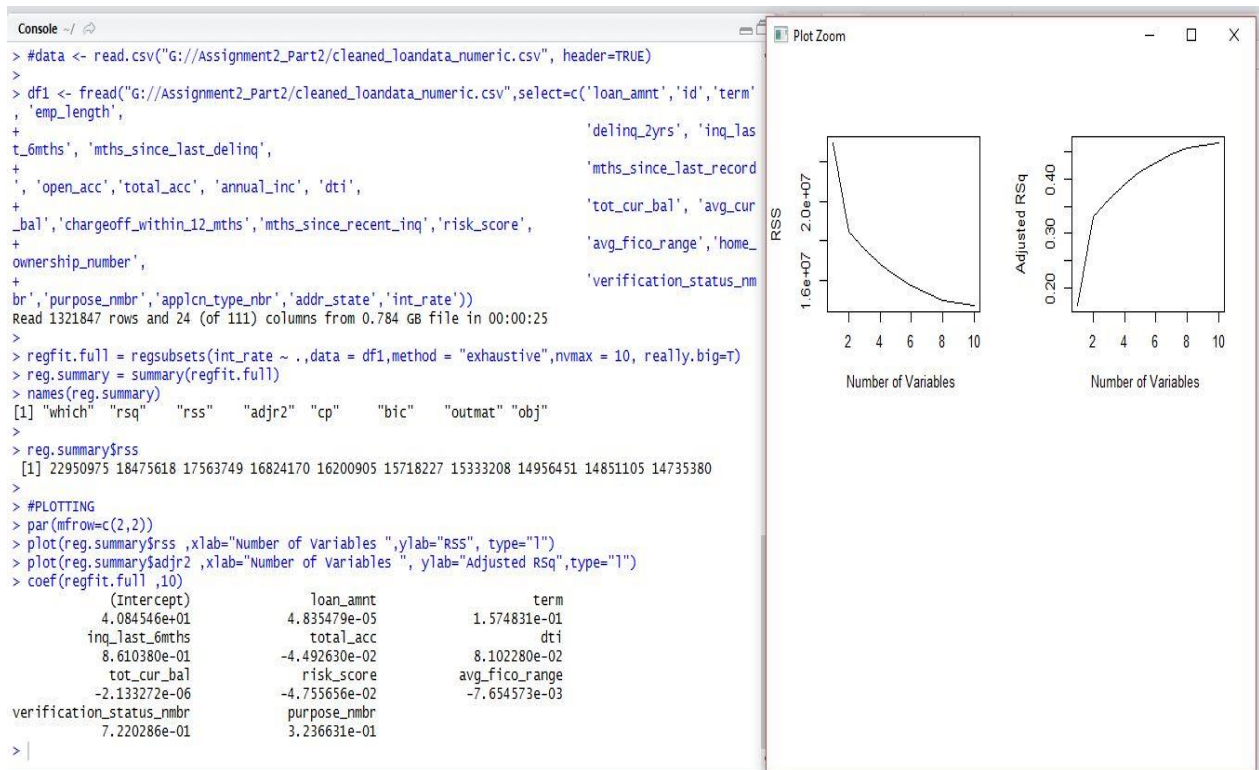

---

 END OF RANDOM FOREST CLASSIFIER
 

---

## FEATURE SELECTION FOR CLUSTERING USING R

- We first factorized the categorical columns in the entire Loan Data Set
- We tried running a feature selection on 45 columns based on correlations. Since we were facing issues with the algorithms, we dilled it down to 25 columns based on correlation, and then ran an exhaustive feature selection on those columns



- We then selected the 10 columns, including the factorized states, id, and interest rate columns for prediction

Loan Amount

Loan Term

Inquiry Last 6 Months

Total number of credit lines

DTI

Total Current Balance of All Accounts

Verification Status Number

Risk Score

Average FICO Range

Purpose Number

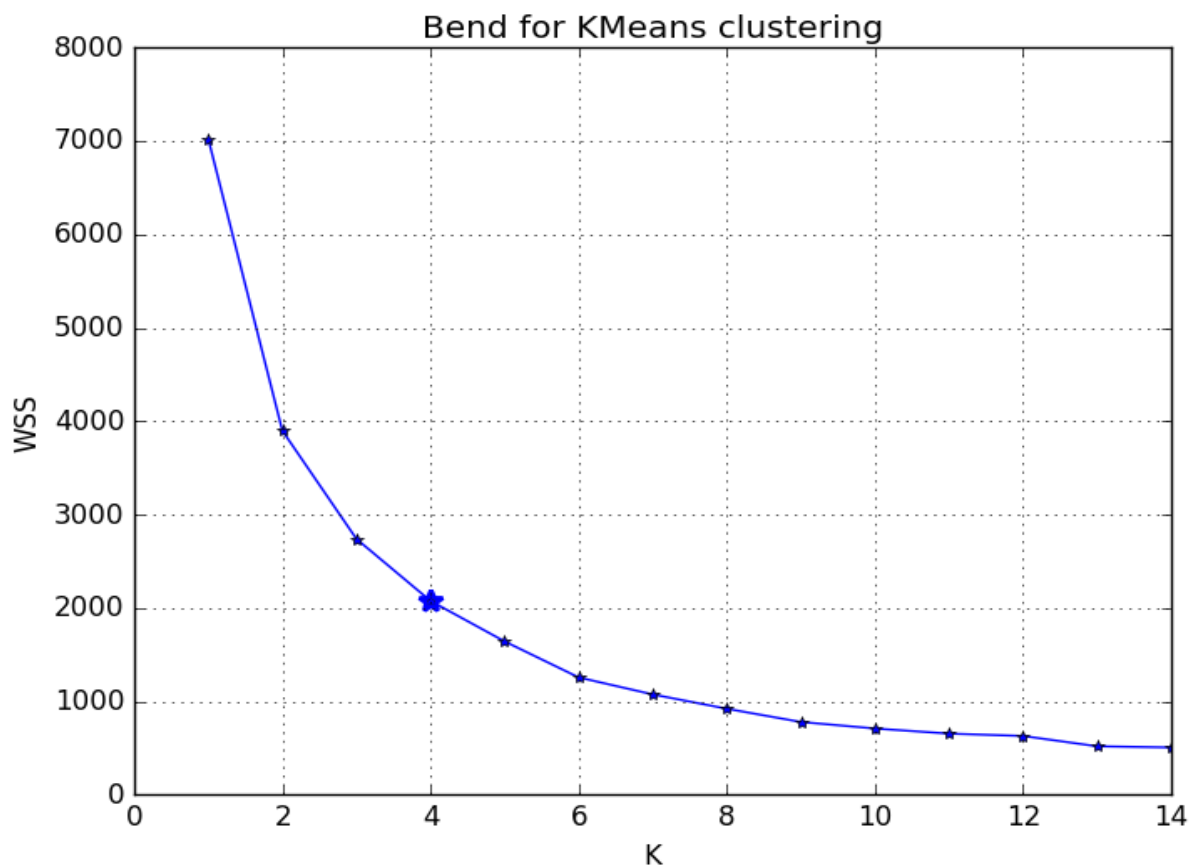
State Address Factors

## CLUSTERING

Language Used: Python

### CLUSTERING USING K-MEANS FOR NUMERIC AND CATEGORICAL VALUES

- For Clustering , we excluded interest rate column, and selected the below 11 columns to find the clusters
- We then produced the Ben graph which suggested we run the K-means Clustering for 4 Clusters



1. We then ran the K-means Clustering algorithm, to detect the centroids, and the labels

```
import numpy as np
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import scipy
from scipy.cluster.vq import kmeans,vq
from scipy.spatial.distance import cdist
import matplotlib.pyplot as plt

k_means=cluster.KMeans(n_clusters=4)

X_tsne = k_means.fit(df1)
centroids = k_means.cluster_centers_

labels = k_means.labels_

#print(centroids)
print(labels)

df1['labels'] = labels

#df1.head()
df1['int_rate'] = loanData['int_rate']
df1['id'] = loanData['id']

downloads_dir_loan = "Data/New"
# Save df1 (kmeans data set with labels) |
df1.to_csv(downloads_dir_loan + "/kmeans_clustering.csv" , sep=',', index = True)
```

2. These values, were appended along with the other selected columns on to a csv file, which was used for prediction

## MANUAL CLUSTERING USING PYTHON

Manual clustering was done on the basis of two derived columns

1. Dti\_Level
2. Credit\_Score\_Level

These two are buckets of Risk\_Score and Dti and are calculated as shown below:

### Dti\_Level:

```
loanData.loc[(loanData['dti'] >= 30), ['Dti_Level']] = 'D'
loanData.loc[(loanData['dti'] >=20) & (loanData['dti']<30), ['Dti_Level']] = 'C'
loanData.loc[(loanData['dti'] >=10) & (loanData['dti']<20), ['Dti_Level']] = 'B'
loanData.loc[(loanData['dti'] >=0) & (loanData['dti']<10), ['Dti_Level']] = 'A'
loanData.loc[(loanData['dti'] <= 0), ['Dti_Level']] = 'Invalid'
```

There are four classes based on the DTI:

Negative DTI (Ignoring this use case)

1. Between 0 and 10 – A
2. Between 10 and 20 – B
3. Between 20 and 30 – C
4. Dti greater than 30

### Credit Score Level:

```
In [2]: loanData['Credit_Score_Level'] = 'Level'
loanData.loc[(loanData['risk_score'] >= 720), ['Credit_Score_Level']] = 'Excellent'
loanData.loc[(loanData['risk_score'] >=680) & (loanData['risk_score']<720), ['Credit_Score_Level']] = 'Good'
loanData.loc[(loanData['risk_score'] >=620) & (loanData['risk_score']<680), ['Credit_Score_Level']] = 'Fair'
loanData.loc[(loanData['risk_score'] >=580) & (loanData['risk_score']<620), ['Credit_Score_Level']] = 'Weak'
loanData.loc[(loanData['risk_score'] >=350) & (loanData['risk_score']<580), ['Credit_Score_Level']] = 'Very Weak'
loanData['Credit_Score_Level'].unique()
```

```
Out[2]: array(['Fair', 'Good', 'Excellent', 'Weak'], dtype=object)
```

There are five buckets based on the risk score:

1. Credit Score > 720 -> Excellent
2. Risk Score between 689 and 720 -> Good
3. Risk Score between 620 and 680 -> Fair
4. Risk Score between 580 and 620 -> Weak
5. Risk Score between 350 and 580 -> Good



The Clusters are then assigned based on the combinations available with these columns as below:

```
: #Manual Clusters with Credit_Score_Level(Good,Fair, Weak, Very Weak) and Dti_Level(A,B,C)
loanData['Manual_Cluster'] = 999

loanData.loc[(loanData['Credit_Score_Level']=='Excellent')&(loanData['Dti_Level']=='A'), ['Manual_Cluster']] = 1
loanData.loc[(loanData['Credit_Score_Level']=='Excellent')&(loanData['Dti_Level']=='B'), ['Manual_Cluster']] = 2
loanData.loc[(loanData['Credit_Score_Level']=='Excellent')&(loanData['Dti_Level']=='C'), ['Manual_Cluster']] = 3

loanData.loc[(loanData['Credit_Score_Level']=='Good')&(loanData['Dti_Level']=='A'), ['Manual_Cluster']] = 1
loanData.loc[(loanData['Credit_Score_Level']=='Good')&(loanData['Dti_Level']=='B'), ['Manual_Cluster']] = 2
loanData.loc[(loanData['Credit_Score_Level']=='Good')&(loanData['Dti_Level']=='C'), ['Manual_Cluster']] = 3

loanData.loc[(loanData['Credit_Score_Level']=='Fair')&(loanData['Dti_Level']=='A'), ['Manual_Cluster']] = 4
loanData.loc[(loanData['Credit_Score_Level']=='Fair')&(loanData['Dti_Level']=='B'), ['Manual_Cluster']] = 5
loanData.loc[(loanData['Credit_Score_Level']=='Fair')&(loanData['Dti_Level']=='C'), ['Manual_Cluster']] = 6

loanData.loc[(loanData['Credit_Score_Level']=='Weak')&(loanData['Dti_Level']=='A'), ['Manual_Cluster']] = 7
loanData.loc[(loanData['Credit_Score_Level']=='Weak')&(loanData['Dti_Level']=='B'), ['Manual_Cluster']] = 8
loanData.loc[(loanData['Credit_Score_Level']=='Weak')&(loanData['Dti_Level']=='C'), ['Manual_Cluster']] = 9

loanData.loc[(loanData['Credit_Score_Level']=='Very Weak')&(loanData['Dti_Level']=='A'), ['Manual_Cluster']] = 10
loanData.loc[(loanData['Credit_Score_Level']=='Very Weak')&(loanData['Dti_Level']=='B'), ['Manual_Cluster']] = 11
loanData.loc[(loanData['Credit_Score_Level']=='Very Weak')&(loanData['Dti_Level']=='C'), ['Manual_Cluster']] = 12
```

Although we have 12 clusters, our data belongs to only the following clusters:

4, 2, 1, 6, 5, 3, 7

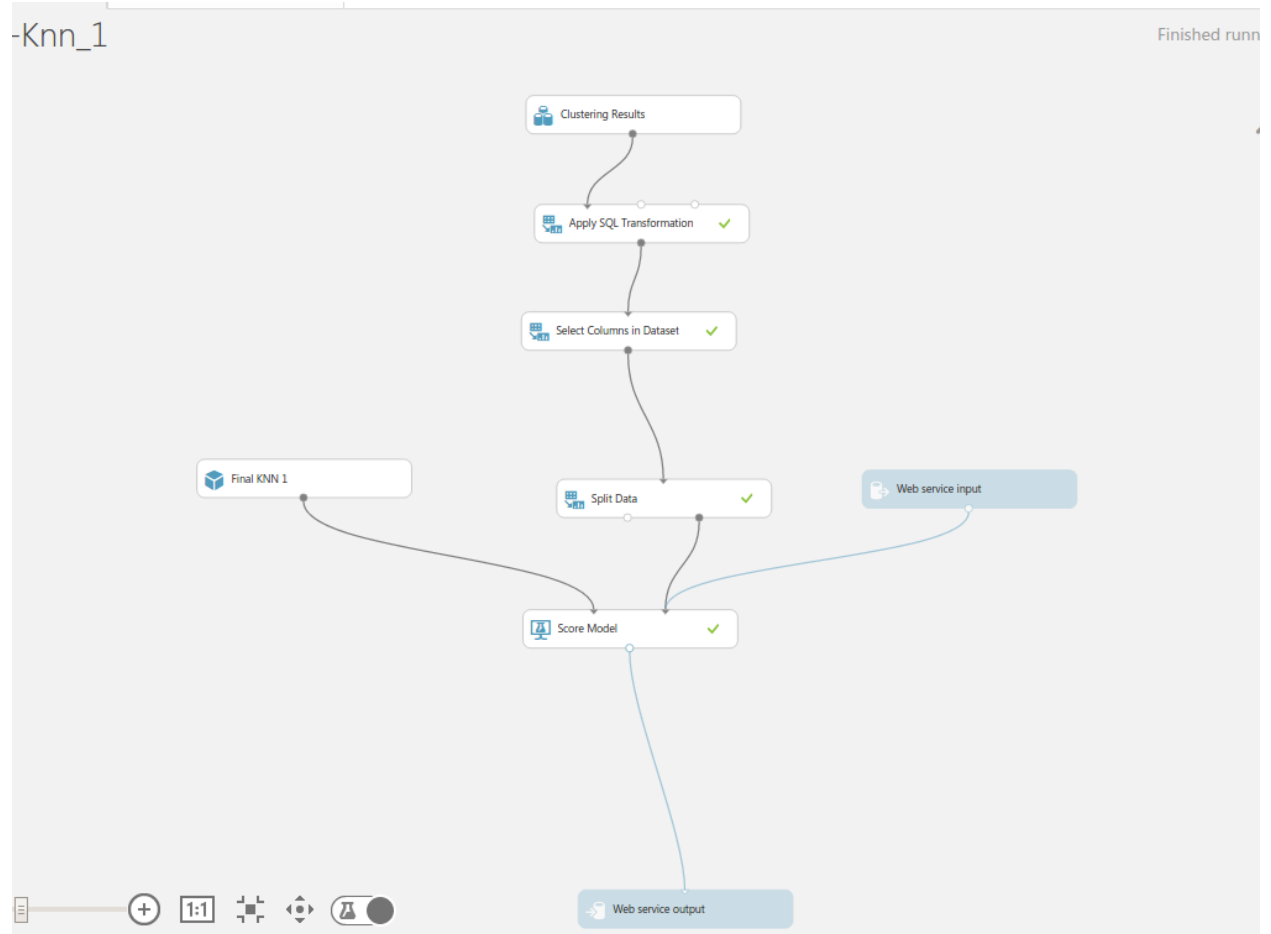
## PREDICTION

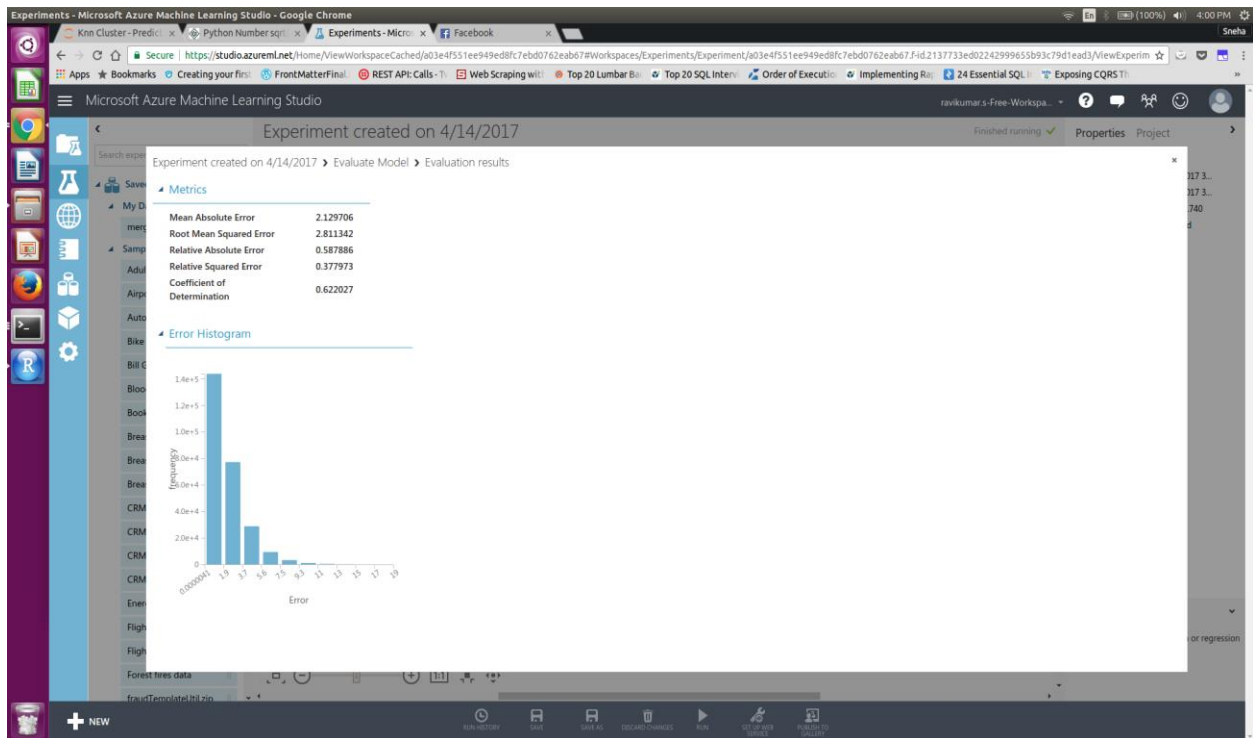
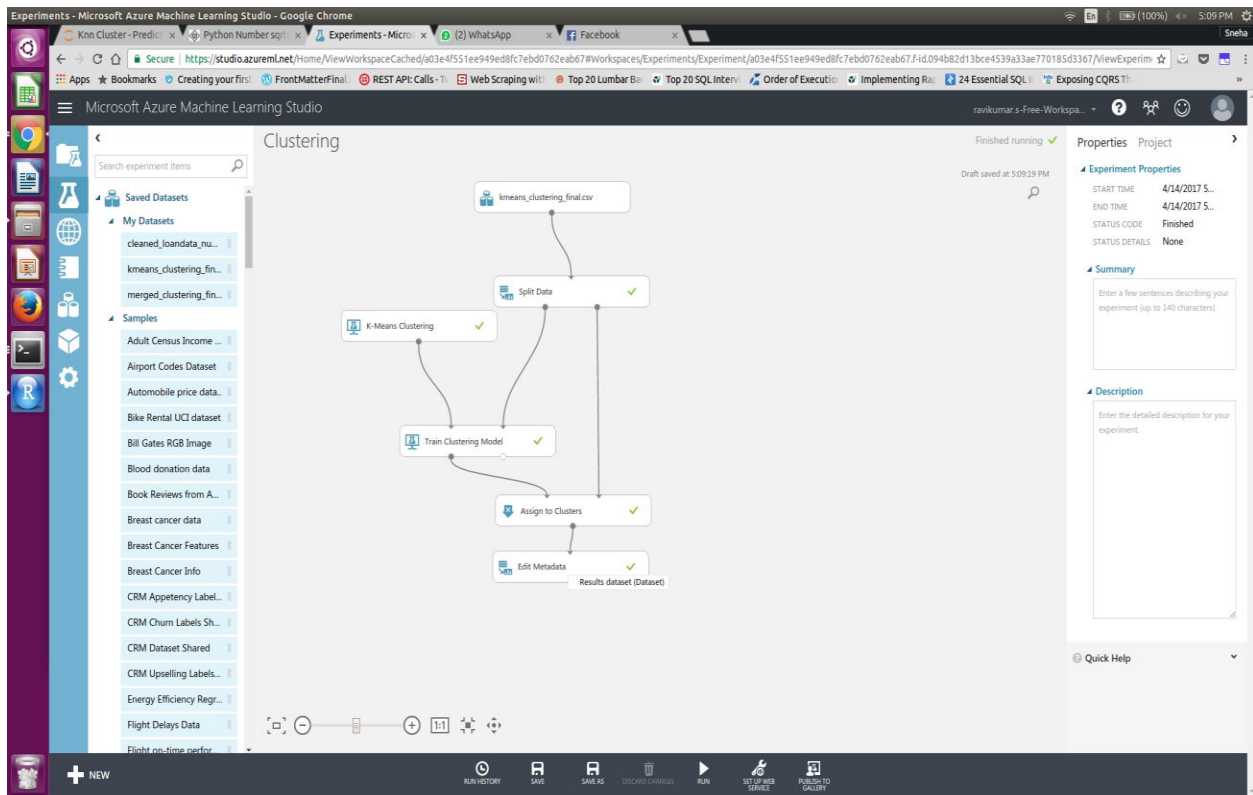
Algorithm	Number of Features Selected With RFE	R-Squared Error	Mean Absolute Error	Mean Squared Error	Median Absolute Error	RMSE	N_estimator	max_depth
Linear Regression	5	0.91	1.07	1.84	0.89	1.35		
Linear Regression	10	0.91	1.07	1.84	0.89	1.35		
Linear Regression	15	0.91	1.06	1.83	0.88	1.35		
Linear Regression	20	0.91	1.06	1.83	0.88	1.354		
Linear Regression	21	0.91	1.05	1.82	0.87	1.34		
Linear Regression	25	0.91	1.05	1.81	0.87	1.34		
Linear Regression	30	0.91	1.05	1.81	0.87	1.34		
Linear Regression	35	0.91	1.05	1.8	0.87	1.34		
Linear Regression	39	0.92	1.01	1.66	0.84	1.29		
Linear Regression	45	0.92	1.01	1.66	0.84	1.28		
Linear Regression	50	0.92	1.01	1.66	0.84	1.28		
KNN Regressor	All	0.61	3.84	24	3.16	4.9		
Random Forest Regressor	45	0.91	1.03	1.69	0.9	1.3	20	7
Random Forest Regressor	30	0.91	1.04	1.7	0.9	1.3	20	7
Random Forest Regressor	24	0.91	1.04	1.7	0.9	1.3	20	7
Random Forest Regressor	24	0.98	1.01	1.64	0.84	1.28	20	none
Random Forest Regressor	45	0.99	0.21	0.21	0.045	0.46	20	5
Neural Network Regressor	All	-83.09	4	15486	3.1	124		

## PREDICTION ON THE ENTIRE DATASET

- We ran the Prediction models using the 4 algorithms, Linear Regression, Random Forest, KNN, and Neural Networks first on the entire dataset
- We got the best results for Random Forest algorithm
- We selected the features using backward, forward, and exhaustive selection and correlations

## PREDICTION MODELS USING MICROSOFT AZURE





## WEB APPLICATION: HOW IT WORKS

- **Web Application Link:** [Sample-env.dvtp6mv4cj.us-east-1.elasticbeanstalk.com](http://sample-env.dvtp6mv4cj.us-east-1.elasticbeanstalk.com)
- The Web application takes in the values from the user, predicts the interest rate, and informs the user if he/she is eligible for a loan or not

The screenshot shows a web application interface for loan eligibility. At the top, there is a dark navigation bar with the text "NEU INFO7390 LENDING CLUB TEAM 7" and four links: "Home", "Report", "R Plots", and "Python Notebooks". Below the navigation bar, a green banner contains the text "Welcome! Check your Loan Eligibility Here!". The main content area is a light gray box with four input fields labeled "Loan Amount:", "Fico Score:", "DTI:", and "Employment Length:". Each field is a white rectangular box. Below these fields is a blue button labeled "Submit Button".

- Once we enter the values above, it takes us to the next page and asks for more information. After classifying the values and approving the amount based on the classification analysis that we've provided above.

Good news!! We can Provide you loan of 7550.0! Now let's see if we can find the interest rate!

**Loan Amount:**

**DTI:**

**FICO Score:**

**Employment Length:**

**Loan Term:**

**Inquiries in last 6 months:**

**Total number of credit lines:**

**Total Current Balance of All Accounts:**

- Once the classification is done, we provide more features, for the prediction of interest rate

Good news!! We can Provide you loan of 7550.0 at 20.58 %

Interest Rate - No Clustering: 20.5829166666667

Interest Rate - K-means Clustering: 19.754375

- If the predicted rates, are good, the loan is approved, and the customer is notified

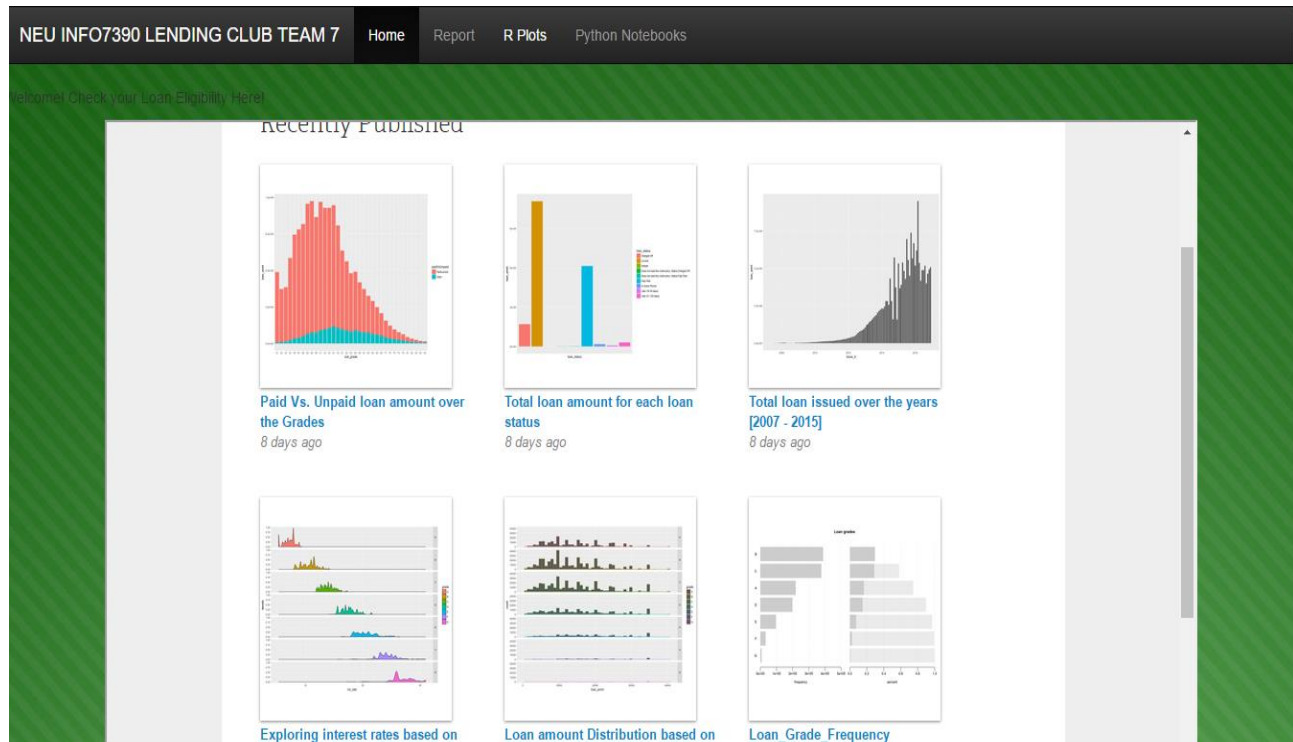


Sorry! We cannot provide you the loan at this time.

- If the classification says that the values are bad, it informs the customer that the loan cannot be provided
- The web application consists of a bunch of apis that take in the values for the entire data set, and predicts the best interest rates and confirms whether or not one is eligible for loan
- We have 4 clusters for the K-means Clustering, 12 Clusters for manual, and one entire data set for which, we have built apis to predict the interest rates using the Random Forest algorithm as explained earlier
- We have a RestAPI that predicts which cluster each record belong to, and accordingly redirects it to the required cluster's prediction model
- We have one RestAPI to predict the interest rate for each cluster



- The site opens up the Data Analysis on the RPlots



- It also displays the Analysis on Python Notebooks



# END OF REPORT