REPORT

INFO 7390 Assignment 2 – Fall 2017 (Lending Club dataset)
Professor - Sri Krishnamurthy

By
Sonali Chaudhari

Problem Statement

You are working at a bank and you are considering investing in Lending club. Since there are no standard models, you are expected to build prediction models that will help you predict the interest rates based on various parameters users would input.

Part 1: Data wrangling and exploratory data analysis

1(a)Data Download and pre-processing:

Data Download:

Steps:

Scraping Data from lending Club

Since the lending club has mode data when signing in, we coded it so there is a login and password. We used requests.session() to access that session we logged into.

```
postUrl = 'https://www.lendingclub.com/info/download-data.action'
payload = {'login_email': "9603sonaligc@gmail.com", 'login_password': "Gopal2145"}
with requests.session() as s:
    loginRequest = s.post(url, data=payload)
    finalUrl = s.get(postUrl)
    linkhtml = finalUrl.text
    soup = BeautifulSoup(linkhtml, "html.parser")
    loan_val = soup.find("div", { "id" : "loanStatsFileNamesJS"}).text
    list_loan_csv = loan_val.split('|')
    if not os.path.exists(newpathloanFinal):
        os.makedirs(newpathloanFinal)
    for value in list_loan_csv:
        url="https://resources.lendingclub.com/"+value
        with urlopen(url) as zipresp:
            with ZipFile(BytesIO(zipresp.read())) as zfile:
                zfile.extractall(newpathloanFinal)
```

We used beautiful soap to find the hidden class with id of loanStatsFileNameJS and same thing but with rejectedStateFileNameJS.

```
def declineloandata():
    direct = os.path.dirname(os.getcwd())
    newpathdeclineloanFinal = direct+'/declineloanstatecsvFinal'
    url = 'https://www.lendingclub.com/account/login.action?'
    postUrl = 'https://www.lendingclub.com/info/download-data.action'
    payload = {'login_email': "9603sonaligc@gmail.com", 'login_password': "Gopal2145"}
    with requests.session() as s:
        loginRequest = s.post(url, data=payload)
        finalUrl = s.get(postUrl)
        linkhtml = finalUrl.text
        soup = BeautifulSoup(linkhtml, "html.parser")
        loan_val = soup.find("div", { "id" : "rejectedLoanStatsFileNamesJS" }).text
        list_loan_csv = loan_val.split('|')
        del list_loan_csv[-1]
        if not os.path.exists(newpathdeclineloanFinal):
            os.makedirs(newpathdeclineloanFinal)
        for value in list_loan_csv:
            url="https://resources.lendingclub.com/"+value
            with urlopen(url) as zipresp:
                with ZipFile(BytesIO(zipresp.read())) as zfile:
                    zfile.extractall(newpathdeclineloanFinal)
```

Data Preprocessing:

Loan Data

Steps:

1. Loaded each programmatically downloaded CSVs in separate dataframe.
2. Concatenated each dataframe in a single dataframe 'loan_data'.

```
In [4]: loan_data.shape
Out[4]: (1524072, 151)
```

3. Going through each column and understanding the meaning of each column through the dictionary provided on the LC website 'LCDataDictionary.xlsx'
4. Getting rid of the columns have more than 70% missing data. During this process we get rid of the column 'id' which has all rows null.

Finding the missing data percentage and storing columns in features with mores than 70% data

```
In [7]: missing_data = loan_df.isnull().sum().to_frame(name='Missing_Count').reset_index()
        missing_data['Missing_percent'] = (missing_data['Missing_Count']/loan_data.shape[0])*100
        missing_data.head()
        missing_data = missing_data.sort_values(by='Missing_percent', axis=0, ascending=True)
        features = missing_data['index'][(missing_data.Missing_percent <30)]
        print('Number of columns after removing missing columns -',len(features))

        Number of columns after removing missing columns - 93

In [8]: loan_df = loan_df[features]

In [9]: del loan_df['id']
        loan_df = loan_df.reset_index()
        loan_df['index'] = loan_df.index +1
        loan_df=loan_df.rename(columns = {'index':'id'})
```

```
In [7]: loan_df.shape
Out[7]: (1524072, 93)
```

5. Out of the 151 columns most of the columns are generated after the loans application procedure has taken place. So these columns leak data from the future. We need columns data that potential customer provided to LC during their application.

Feature Engineering

6. Created a list 'f' a features that are important for and received during initial borrower's application.
7. Created a dataframe 'df' consisting of all the features in the list 'f' and generated a column 'id' for uniquely identifying each entry.

```
In [16]: # df is the main dataframe required for working
         df = s[f]
```

```
s['index'] = s.index +1
s=s.rename(columns = {'index':'id'})
```

```
'annual_inc',
'dti_joint',
'annual_inc_joint',
'grade',
'sub_grade',
'int_rate',
'loan_amnt',
'issue_d',
'purpose',
'addr_state',
'zip_code']
```

8. Retrieved the count of nulls in each column in ‘df” to deal with missing data in every column.

```
In [49]: df.isnull().sum()
Out[49]: id                      0
         application_type        1
         fico_range_low          1
         fico_range_high         1
         emp_length              1
         dti                   174
         annual_inc              5
         dti_joint         1503556
         annual_inc_joint  1503552
         grade                   1
         sub_grade               1
         int_rate                1
         loan_amnt               1
         issue_d                 1
         purpose                 1
         addr_state              1
         zip_code                2
         dtype: int64
```

9. Getting rid of the row having most of the columns null.

```
In [19]: df = df[df.application_type.notnull()]
```

10. Getting the dataframe to have consistent formats for each column entry and logically replacing the NaN values for some categorial columns

```
In [48]: df.head()
```

Out[48]:

id	application_type	fico_range_low	fico_range_high	emp_length	dti	annual_inc	dti_joint	annual_inc_joint	grade	sub_grade	int_rate	loan_a
1	Individual	735.0	739.0	10+ years	27.65	24000.0	NaN	NaN	B	B2	10.65%	5000.0
2	Individual	740.0	744.0	< 1 year	1.00	30000.0	NaN	NaN	C	C4	15.27%	2500.0
3	Individual	735.0	739.0	10+ years	8.72	12252.0	NaN	NaN	C	C5	15.96%	2400.0
4	Individual	690.0	694.0	10+ years	20.00	49200.0	NaN	NaN	C	C1	13.49%	10000.(
5	Individual	695.0	699.0	1 year	17.94	80000.0	NaN	NaN	B	B5	12.69%	3000.0

```
In [63]: df['dti_joint'] = np.where(df['application_type']=='Individual',0,df['dti_joint'])

In [65]: df['annual_inc_joint'] = np.where(df['application_type']=='Individual',0,df['annual_inc_joint'])
```

dti_joint filling

```
In [69]: part = df[['dti_joint', 'sub_grade','id']][df.application_type=='Joint App']
         Mode = df['dti_joint'].mode()[0]
         a = list(df['sub_grade'].unique())

         for c in a:
             t = part[part.sub_grade == c]
             if( t.dti_joint.isnull().sum()==0):
                     break
             else:
                     if(t.shape == t[t.isnull().any(axis=1)].shape ):
                         t['dti_joint'].fillna(Mode, inplace=True)
                     else:
                         c = t['dti_joint'].mode()[0]
                         t['dti_joint'].fillna(c, inplace=True)
                     part.loc[part.id.isin(t.id), ['dti_joint']] = t[['dti_joint']]
         part['dti_joint'].fillna(part['dti_joint'].mode()[0], inplace = True)
         df.loc[df.id.isin(part.id), ['dti_joint']] = part[['dti_joint']]
```

1. The columns like emp_length, int_rates, zipcode, dti… are of type object and has no consistent format
2. Replaced dti_joint, annual_inc_joint null values to '0' for the application_type

annual_inc

```
In [74]: part = df[['id','sub_grade', 'annual_inc']]
         Mode = df['annual_inc'].mode()[0]
         a = list(df['sub_grade'].unique())

         for c in a:
             t = part[part.sub_grade == c]
             if( t.annual_inc.isnull().sum()==0):
                     break
             else:
                     if(t.shape == t[t.isnull().any(axis=1)].shape ):
                         t['annual_inc'].fillna(Mode, inplace=True)
                     else:
                         c = t['annual_inc'].mode()[0]
                         t['annual_inc'].fillna(c, inplace=True)
                     part.loc[part.id.isin(t.id), ['annual_inc']] = t[['annual_inc']]
         part['annual_inc'].fillna(part['annual_inc'].mode()[0], inplace = True)
         df.loc[df.id.isin(part.id), ['annual_inc']] = part[['annual_inc']]
```

'Individual' since those application has no co-borrower and hence dti_joint, annual_inc_joint becomes 0.

3. Filled the remaining null values for annual_inc with the mode annual_inc value in a particular sub_grade column that the row belongs to.
4. Similarly for dti_joint column
5. There are columns 'fico_range_high' and 'fico_range_low'; so generated a new column 'fico_avg' having the average of both the column values
6. Change the datatype of necessary columns and adding new column approval having value 1 for the loan data.

Fico columns

```
In [ ]: loan_df['approval'] =1
        loan_df=loan_df.rename(columns = {'fico_avg':'fico'})
        loan_df=loan_df.rename(columns = {'addr_state':'State'})

In [ ]: loan_df.to_csv('/Users/sonalichaudhari/Desktop/Loan.csv', index=False)
```

Decline Data

Steps:

1. Loaded the csvs in each data frames and then concatenating it one data frames.
2. Clean the data
3. Changing the column names and datatype
4. Adding new column ‘approval’ having values ‘0’ for the decline data
5. Exporting to Csv

```
def concatDeclineloandatacsv():
    #direct = os.path.dirname(os.getcwd())
    newpathdeclineloanFinal = '/Users/sonalichaudhari/Desktop/Decline'
    d1 = pd.read_csv(newpathdeclineloanFinal+'/RejectStatsA.csv', header=1, skipfooter=4, engine='python')
    d2 = pd.read_csv(newpathdeclineloanFinal+'/RejectStatsB.csv', header=1, skipfooter=4, engine='python')
    d3 = pd.read_csv(newpathdeclineloanFinal+'/RejectStatsD.csv', header=1, skipfooter=4, engine='python')
    d4 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2016Q1.csv', header=1, skipfooter=4, engine='python')
    d5 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2016Q2.csv', header=1, skipfooter=4, engine='python')
    d6 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2016Q3.csv', header=1, skipfooter=4, engine='python')
    d7 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2016Q4.csv', header=1, skipfooter=4, engine='python')
    d8 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2017Q1.csv', header=1, skipfooter=4, engine='python')
    d9 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2017Q2.csv', header=1, skipfooter=4, engine='python')
    d10 = pd.read_csv(newpathdeclineloanFinal+'/RejectStats_2017Q3.csv', header=1, skipfooter=4, engine='python')
    concatdata = pd.concat([d1, d2, d3, d4, d5, d6, d7, d8, d9, d10])
    return concatdata
```

```
concatDataReject = concatDeclineloandatacsv()
```

```
concatDataReject.shape
```

```
(16131432, 9)
```

```
concatDataReject.head()
```

	Amount Requested	Application Date	Loan Title	Risk_Score	Debt-To-Income Ratio	Zip Code	State	Employment Length	Policy Code
0	1000.0	2007-05-26	Wedding Covered but No Honeymoon	693.0	10%	481xx	NM	4 years	0.0
1	1000.0	2007-05-26	Consolidating Debt	703.0	10%	010xx	MA	< 1 year	0.0
2	11000.0	2007-05-27	Want to consolidate my debt	715.0	10%	212xx	MD	1 year	0.0
3	6000.0	2007-05-27	waksman	698.0	38.64%	017xx	MA	< 1 year	0.0
4	1500.0	2007-05-27	mdrigo	509.0	9.43%	209xx	MD	< 1 year	0.0

Cleaning

```
# Risk Score
concatDataReject= concatDataReject[concatDataReject.Risk_Score.notnull()]
concatDataReject[concatDataReject.Risk_Score<300]

# State
concatDataReject = concatDataReject[concatDataReject.State.notnull()]
concatDataReject['State'].fillna('Other',inplace = True)

# ID
concatDataReject['id'] = concatDataReject.index + 1

# DTI
concatDataReject['Debt-To-Income Ratio'] = concatDataReject['Debt-To-Income Ratio'].astype(str).map(lambda x: x.r
concatDataReject['Debt-To-Income Ratio'] = concatDataReject['Debt-To-Income Ratio'].astype(float)

# Application Date
concatDataReject['Year'] = concatDataReject['Application Date'].astype(str).map(lambda x: int(x.split('-')[0]))
concatDataReject['Month'] = concatDataReject['Application Date'].astype(str).map(lambda x: x.split('-')[1])
concatDataReject['Day'] = concatDataReject['Application Date'].astype(str).map(lambda x: x.split('-')[2])
concatDataReject['Month']=concatDataReject['Month'].astype(int)
concatDataReject['Day']=concatDataReject['Day'].astype(int)

# Loan Title
concatDataReject['Loan Title'].fillna('None', inplace=True)

#Zipcode
concatDataReject['Zip Code'] = concatDataReject['Zip Code'].astype(str).map(lambda x: x.rstrip('xx'))
#missingzipcode()

# Policy Code
Mode = concatDataReject['Policy Code'].mode()[0]
concatDataReject['Policy Code'].fillna(2, inplace=True)


#concatDataReject['State'] = concatDataReject['State'].fillna('NA',inplace= True)

# Employement Length
concatDataReject['Employment Length']=np.where(concatDataReject['Employment Length']=='10+ years','10',concatData
concatDataReject['Employment Length']=np.where(concatDataReject['Employment Length']=='< 1 year','0',concatDataRe
concatDataReject['Employment Length']=np.where(concatDataReject['Employment Length']=='1 year','1',concatDataReje
concatDataReject['Employment Length'] = concatDataReject['Employment Length'].map(lambda x: x.rstrip('years'))
concatDataReject['Employment Length'] = concatDataReject['Employment Length'].map(lambda x: x.strip())
concatDataReject.loc[concatDataReject['Employment Length']=='n/',['Employment Length']]=0
concatDataReject['Employment Length']= concatDataReject['Employment Length'].astype(np.int64)

concatDataReject['approval']=0
```

Luigi

We have created 2 files. One for decline data and one for loan data.

```
# Renaming
concatDataReject=concatDataReject.rename(columns = {'Amount Requested':'loan_amnt'})
concatDataReject=concatDataReject.rename(columns = {'Application Date':'application_date'})
concatDataReject=concatDataReject.rename(columns = {'Loan Title':'purpose'})
concatDataReject=concatDataReject.rename(columns = {'Zip Code':'zip_code'})
concatDataReject=concatDataReject.rename(columns = {'Debt-To-Income Ratio':'dti'})
concatDataReject=concatDataReject.rename(columns = {'Employment Length':'emp_length'})
concatDataReject=concatDataReject.rename(columns = {'Risk_Score':'fico'})
concatDataReject=concatDataReject.rename(columns = {'Application Date':'application_date'})
```

```
→  luigi python luigiscriptfinal.py --local-scheduler MissingandFeatureSelectDeclineData
/Users/sonalichaudhari/anaconda/lib/python3.5/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: T
ch all the refactored classes and functions are moved. Also note that the interface of the new CV iterators are d
  "This module will be removed in 0.20.", DeprecationWarning)
DEBUG: Checking if MissingandFeatureSelectDeclineData() is complete
DEBUG: Checking if ScrapeDeclineData() is complete
INFO: Informed scheduler that task   MissingandFeatureSelectDeclineData__99914b932b   has status   PENDING
INFO: Informed scheduler that task   ScrapeDeclineData__99914b932b   has status   PENDING
INFO: Done scheduling tasks
INFO: Running Worker with 1 processes
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 2
INFO: [pid 8297] Worker Worker(salt=552557791, workers=1, host=Sonalis-MacBook-Pro.local, username=sonalichaudhar
INFO: [pid 8297] Worker Worker(salt=552557791, workers=1, host=Sonalis-MacBook-Pro.local, username=sonalichaudhar
DEBUG: 1 running tasks, waiting for next task to finish
INFO: Informed scheduler that task   ScrapeDeclineData__99914b932b   has status   DONE
DEBUG: Asking scheduler for work...
DEBUG: Pending tasks: 1
INFO: [pid 8297] Worker Worker(salt=552557791, workers=1, host=Sonalis-MacBook-Pro.local, username=sonalichaudhar
INFO: [pid 8297] Worker Worker(salt=552557791, workers=1, host=Sonalis-MacBook-Pro.local, username=sonalichaudhar
DEBUG: 1 running tasks, waiting for next task to finish
INFO: Informed scheduler that task   MissingandFeatureSelectDeclineData__99914b932b   has status   DONE
DEBUG: Asking scheduler for work...
DEBUG: Done
DEBUG: There are no more tasks to run at this time
INFO: Worker Worker(salt=552557791, workers=1, host=Sonalis-MacBook-Pro.local, username=sonalichaudhari, pid=8297
INFO:
===== Luigi Execution Summary =====

Scheduled 2 tasks of which:
* 2 ran successfully:
    - 1 MissingandFeatureSelectDeclineData()
    - 1 ScrapeDeclineData()

This progress looks :) because there were no failed tasks or missing external dependencies
```

1(b)Exploratory Data analysis:

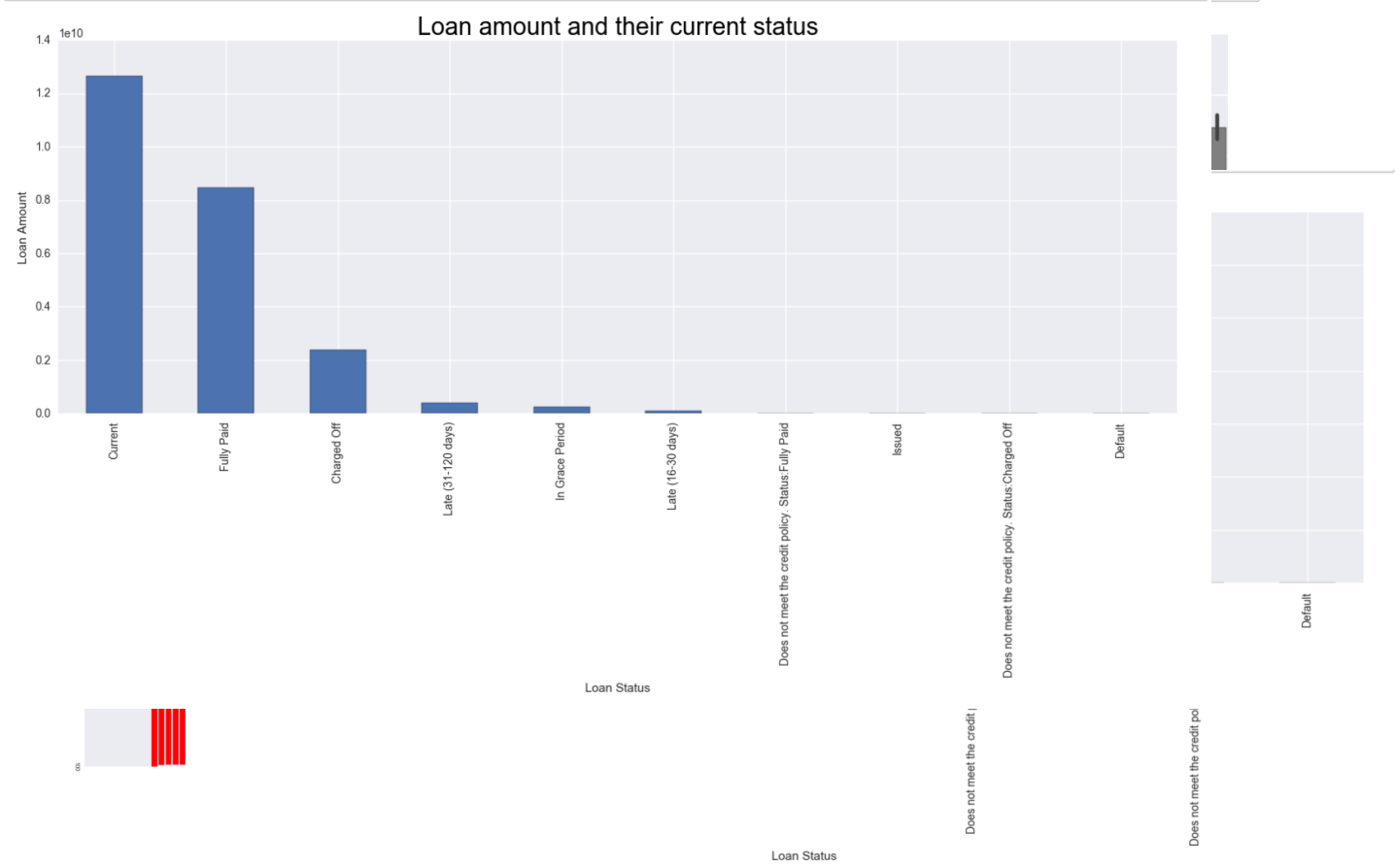
We performed exploratory data analysis the loan data and decline data separately and few EDAs for combine data.

Loan data

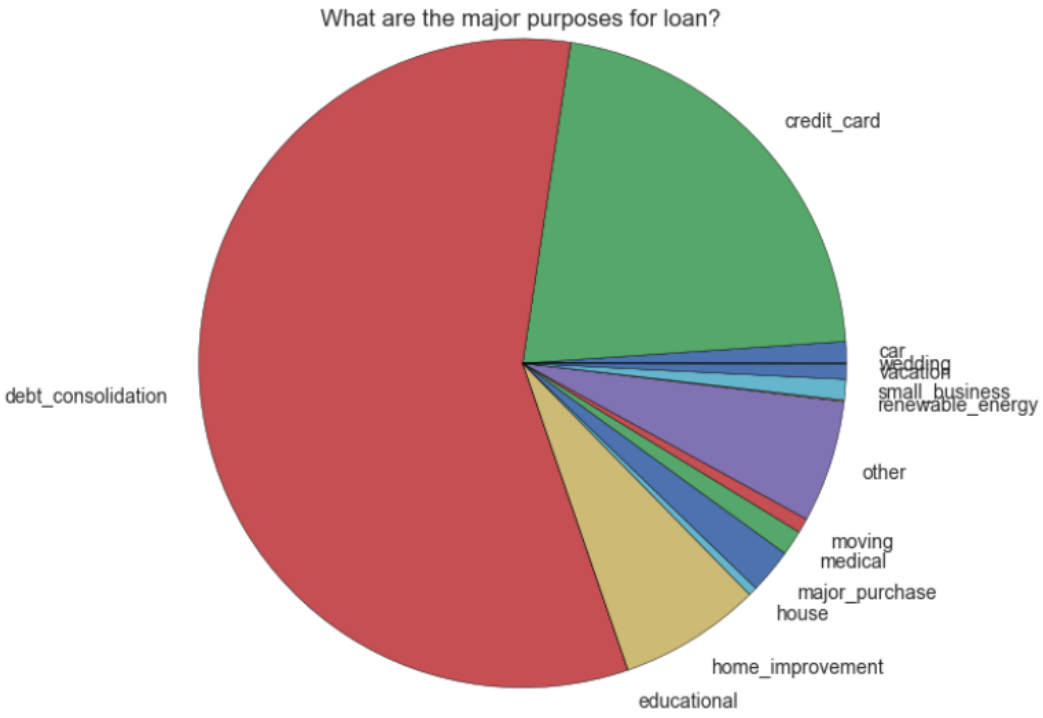
1. Missing data Analysis
2. How many loans have been approved per grade and sub grade?
3. Distribution of Interest Rate

4. Distribution of Loan Amount
5. Loans Granted per State
6. Zico high and fido low analysis

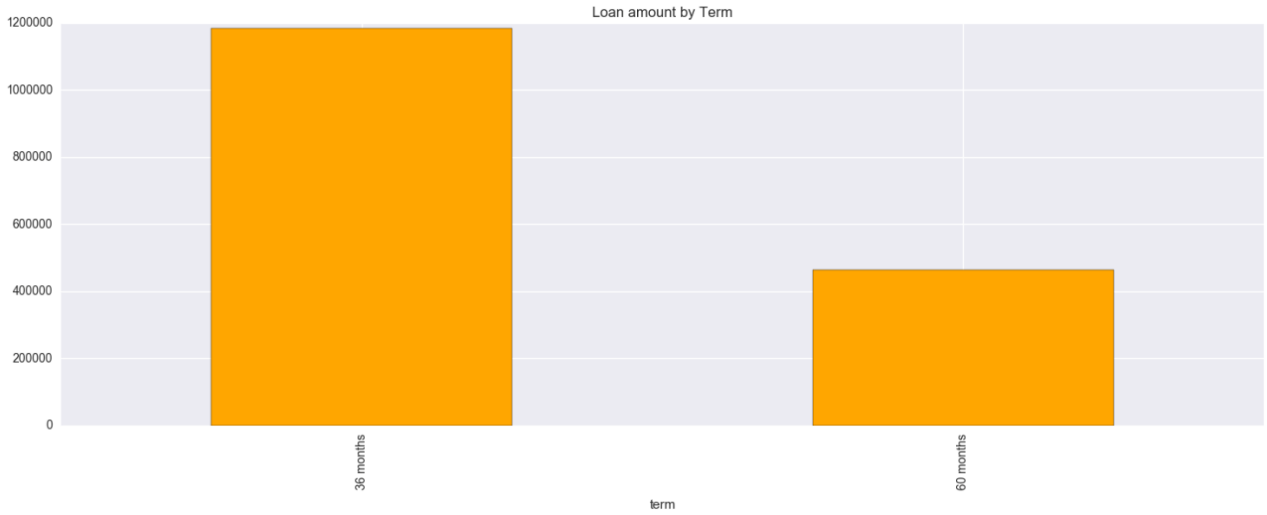
7. Loan Amount by loan status



8. Distribution of Loan amount over purpose

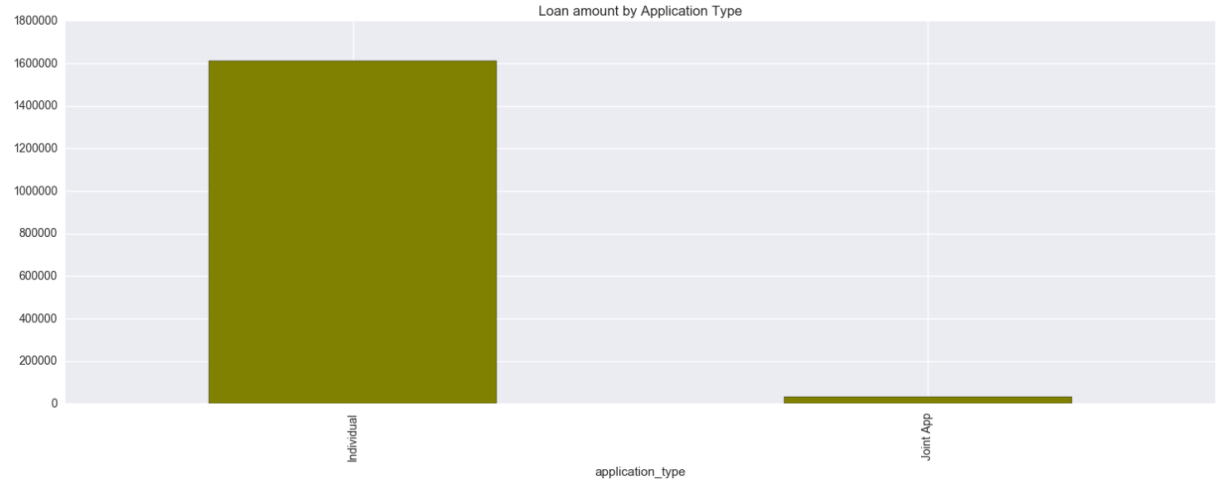


9. Distribution of Loan amount over term
10.

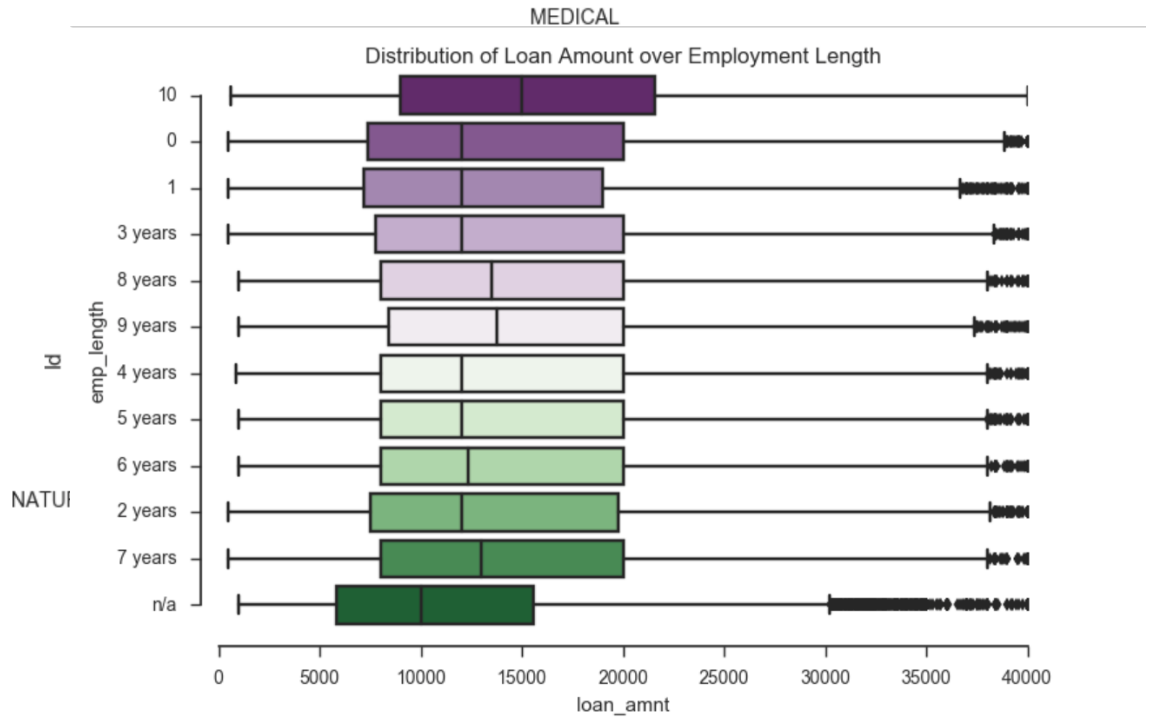


11. Distribution of Loan amount over application type

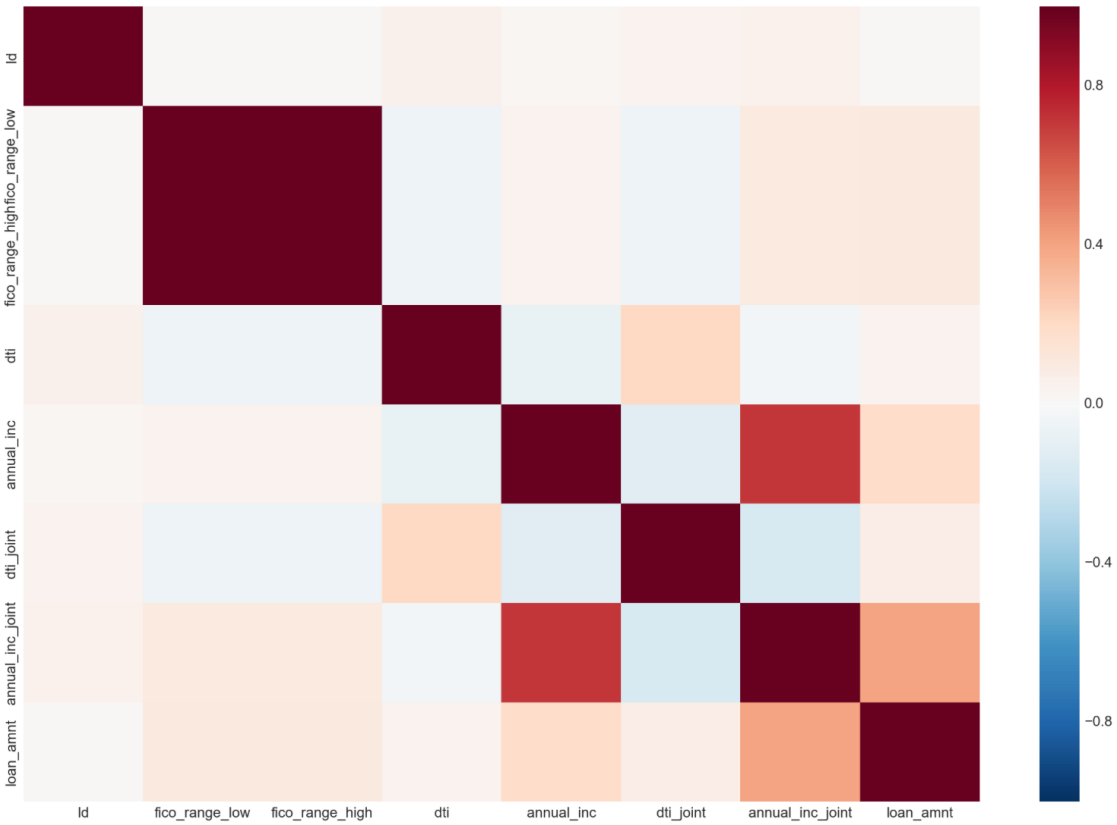
```
application_type
Individual    1612258
Joint App       34514
Name: application_type, dtype: int64
```



12. Loan Amount count per Hardship Reason

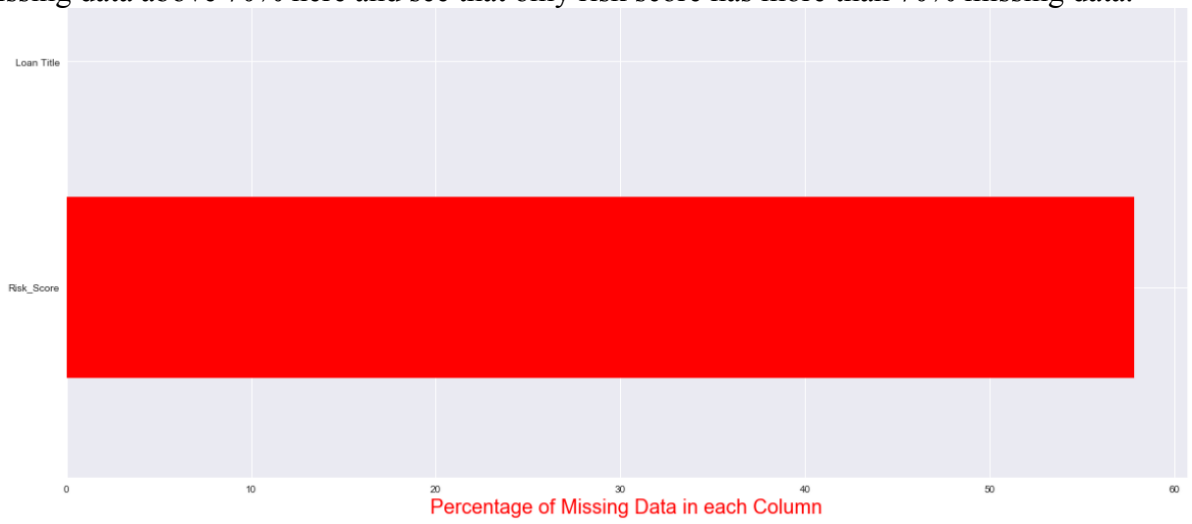


13. Distribution of Loan Amount over Employment Length
14. Heat Map of columns details provided during application for Loan



EDA for Rejected Loans

Looking at the rejected data, it is clear that there are only 9 columns. We are finding all the missing data above 70% here and see that only risk score has more than 70% missing data.



We have then dealt with looking at description of each of important columns to see the min, max, mean , and mode.

```
decline_data['Amount Requested'].describe()
```

```
count    1.613143e+07
mean     1.327537e+04
std      1.584008e+04
min      0.000000e+00
25%      4.200000e+03
50%      1.000000e+04
75%      2.000000e+04
max      1.400000e+06
Name: Amount Requested, dtype: float64
```

```
decline_data['Loan Title'].describe()
```

```
count                16130137
unique                  73929
top       debt_consolidation
freq                  5809883
Name: Loan Title, dtype: object
```

```
decline_data['Debt-To-Income Ratio'].describe()
```

```
count     16131432
unique       97768
top             -1%
freq       1106449
Name: Debt-To-Income Ratio, dtype: object
```

```
decline_data['Zip Code'].describe()
```

```
count     16131139
unique        1001
top          112xx
freq        161823
Name: Zip Code, dtype: object
```

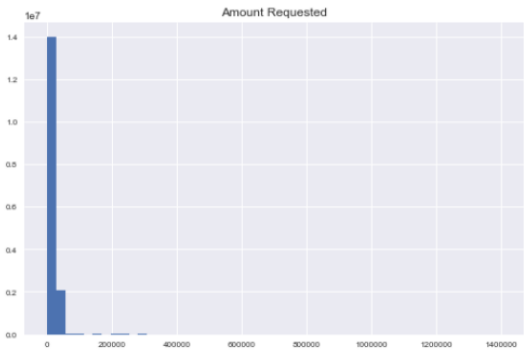
```
decline_data['Employment Length'].describe()
```

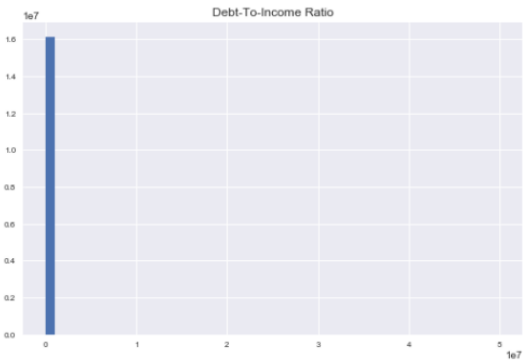
```
count     16131432
unique          12
top       < 1 year
freq      12479825
Name: Employment Length, dtype: object
```

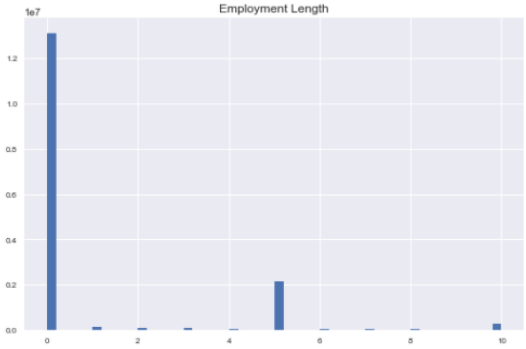
```
decline_data['Policy Code'].describe()
```

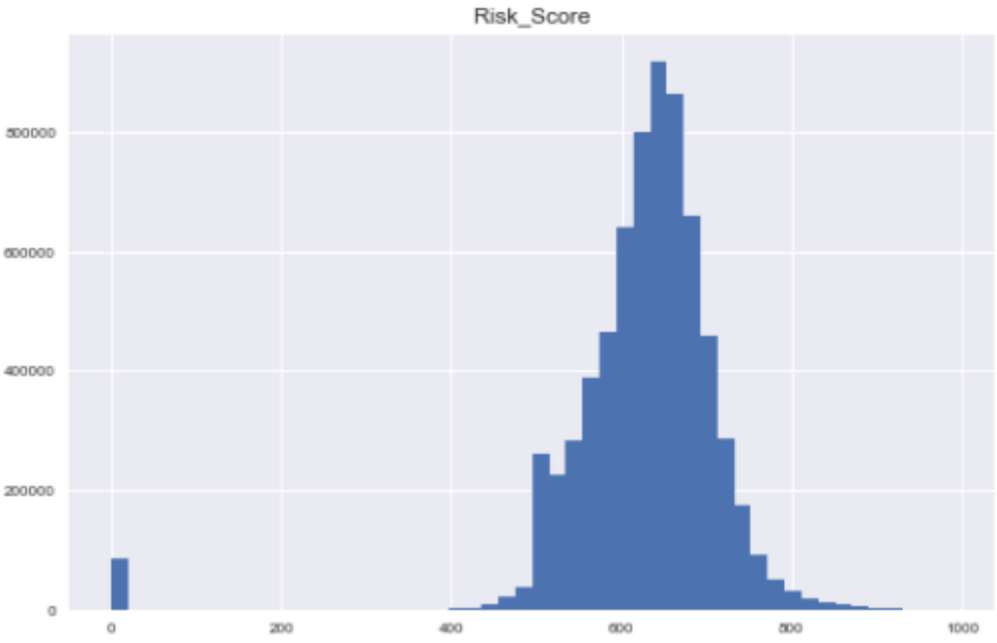
```
count    1.613051e+07
mean     5.593126e-03
std      1.056171e-01
min      0.000000e+00
25%      0.000000e+00
50%      0.000000e+00
75%      0.000000e+00
max      2.000000e+00
Name: Policy Code, dtype: float64
```

We then verified and checked the distribution of the columns and the data to see if there are a lot of outliers and how skewed the data was.

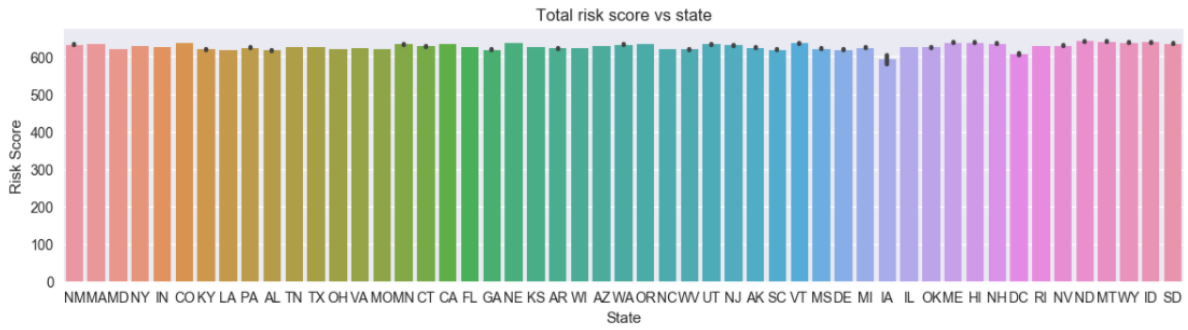




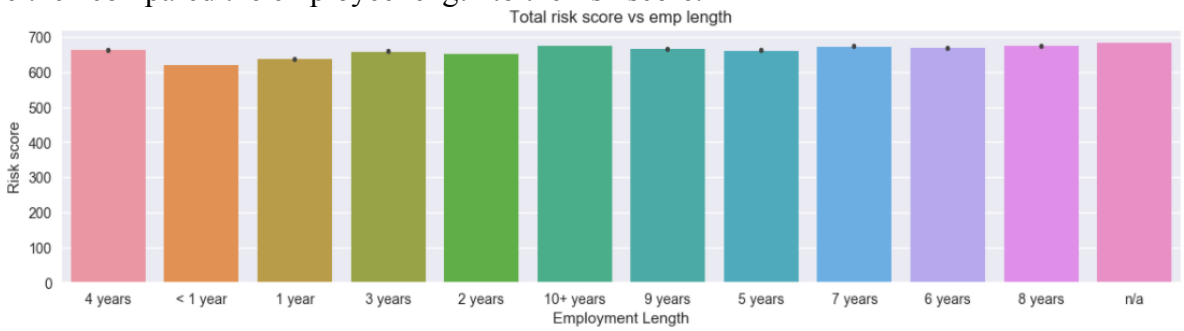




We compared the state and to the risk score



We then compared the employee length to the risk score.



We also verified all the unique values of the loan title to see the types of loans people were taking out.

```
'Wedding Covered but No Honeymoon', 'Consolidating Debt',
'Want to consolidate my debt', ..., 'dougie03', 'freeup',
'Business Advertising Loan'], dtype=object)
```

1(c)Luigi Scripts:

Luigi makes it easier to automate the pipeline.
I created a py file. The luigi script has 2 files and 2 classes in each it has a run and output methods. One class is for rejected and another for loan accepted.

In Loan Luigi we have one class for parsing web and another for dealing with missing data and feature selection.

Part 2: Building and evaluating models

2(a): Classification

- For classification, we combined the loan and decline data keeping only the columns in both the datasets. We have generated a flag column named ‘approval’ during the cleaning process.

approval ->1 Loan data

approval ->0 Decline data

- We create a dataframe having equal number of rows(1640753) from loan and decline data.
- Features we are dealing with for classification:

['id','loan_amnt', 'purpose', 'fico', 'dti','zip_code', 'State', 'emp_length',

'Year', 'Month', ‘approval']]

- Then we spilt the data into train and test having 70/30 split.

Cross Validation for splitting the data in train and test sets

In [51]:
```
# columns to consider
cols = ['loan_amnt','emp_length','risk_score','dti']
```

In [52]:
```
X = com[cols]
y = com['approval']

#Flatten y into a 1-D array
y = np.ravel(y)
```

In [53]:
```
# Splitting the data
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y,test_size=0.30)
```

In [55]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
# Converting the categorical data to numeric
X_train['pur'] = label_encoder.fit_transform(X_train['purpose'])
X_test['pur'] = label_encoder.fit_transform(X_test['purpose'])

X_train['state'] = label_encoder.fit_transform(X_train['State'])
X_test['state'] = label_encoder.fit_transform(X_test['State'])
```

In [56]:
```
X_train = X_train._get_numeric_data()
X_test = X_test._get_numeric_data()
```

Models Implemented

We implement **Logistic regression, Random Forest, Neural Network** models algorithms.

Random Forest worked best for the classification.

Logistic Regression:

Random Forest:

```
#Compute FPR and TPR
from sklearn.metrics import roc_curve
```

```
# Creating the model
nn = MLPClassifier(hidden_layer_sizes=(50,20,5),max_iter=20,verbose=False)
nn.fit(X_train, y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(50, 20, 5), learning_rate='constant',
       learning_rate_init=0.001, max_iter=20, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

```
accuracy = nn.score(X_train, y_train)
print(accuracy)
```

```
0.903920848182
```

```
#Run the model on the test set
y_pred = nn.predict(X_test)
```

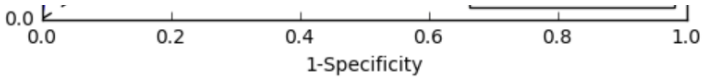
```
y_pred
```

```
array([0, 0, 0, ..., 1, 0, 1])
```

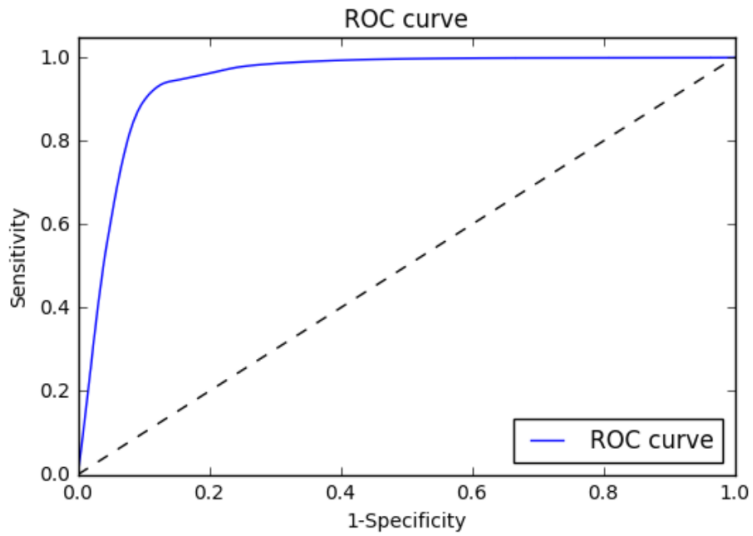
```
# Predict the values for test and train using the model
y_test_predicted = nn.predict(X_test)
y_train_predicted= nn.predict(X_train)
```

```
#Compute confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[429825  61969]
 [ 32151 460507]]
```




```
#Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```



Neural Networks:

2(b): Clustering

We have segmented our loan data broadly in ways:

1. No cluster(Zero Cluster)
2. Manually
3. Clustering Algorithm(K-Means)

NO Cluster:

We loaded the loan data as is.

MANUALLY:

We cluster data based on the fico score in to four clusters. And generated a new column ‘cluster’ for loan data frame. Based on the fido score in each row of loan data we fill in the cluster column with the corresponding cluster name. Then exporting each cluster to a separate CSV.

```
k1 = df[df.cluster=='K1']
del k1['cluster']
k1.to_csv('/Users/sonalichaudhari/Desktop/ADS_A_2/Cluster/Manual_1.csv',index=False)
```

```
k2 = df[df.cluster=='K2']
del k2['cluster']
k2.to_csv('/Users/sonalichaudhari/Desktop/ADS_A_2/Cluster/Manual_2.csv',index=False)
```

```
k3 = df[df.cluster=='K3']
del k3['cluster']
k3.to_csv('/Users/sonalichaudhari/Desktop/ADS_A_2/Cluster/Manual_3.csv',index=False)
```

```
k4 = df[df.cluster=='K4']
del k4['cluster']
k4.to_csv('/Users/sonalichaudhari/Desktop/ADS_A_2/Cluster/Manual_4.csv',index=False)
```

```
def clustering(fico):
    cluster_name = ''
    if fico>790.0:
        cluster_name ='K1'
    elif ((fico <= 790.0) & (fico>750.0)):
        cluster_name = 'K2'
    elif ((fico <=750.0) & (fico>700.0)):
        cluster_name ='K3'
    elif ((fico <=700.0) & (fico>660.0)):
        cluster_name= 'K4'
    return cluster_name
```

```
df['cluster'] = df['fico'].astype(float).map(lambda x: clustering(x))
```

K- Means Clustering:

Here we cluster into 8 different clusters and also we are actually dealing with same features as manual cluster. We have normalized data before creating cluster.

```
k_mean.shape
```

```
(1640757, 27)
```

```
k_mean.dtypes
```

```
norm_data = preprocessing.normalize(k_mean)
```

```
norm_data
```

```
array([[  1.12735567e-03,   9.78536570e-01,   3.00492272e-02, ...,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  3.32080402e-05,   9.96241206e-01,   2.46403658e-02, ...,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  6.97229400e-04,   9.79639290e-01,   5.89286775e-02, ...,
          7.99575000e-05,   0.00000000e+00,   0.00000000e+00],
       ...,
       [  3.07366011e-04,   9.06149797e-01,   8.54197209e-03, ...,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  1.35808005e-04,   9.59612419e-01,   6.14151948e-03, ...,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00],
       [  5.16259123e-04,   9.21891291e-01,   1.39717747e-02, ...,
          0.00000000e+00,   0.00000000e+00,   0.00000000e+00]])
```

```
k_cluster=cluster.KMeans(n_clusters=8)
```

```
k_cluster.fit(norm_data)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=8, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

```
k_cluster.inertia_
```

```
843.86228211237938
```

```
|: k_cluster=cluster.KMeans(n_clusters=8)
```

```
|: k_cluster.fit(norm_data)
```

```
|: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
      n_clusters=8, n_init=10, n_jobs=1, precompute_distances='auto',
      random_state=None, tol=0.0001, verbose=0)
```

```
|: k_cluster.inertia_
```

```
|: 843.86228211237938
```

```
|: cluster_label = k_cluster.labels_
```

We then create a pkl file which exports the model. Also, we validate the code based on the grade to see if all clusters return data.

```
with open('kmeanCluster.pkl', 'wb') as f:
    joblib.dump(k_cluster, f)
```

```
loan_df['kmean'] = cluster_label
```

```
pd.crosstab(loan_df.grade,loan_df.kmean)
```

kmean	0	1	2	3	4	5	6	7
grade								
A	62287	15675	44518	49853	258	27988	6121	65114
B	102494	40864	68107	92256	846	62852	18927	97492
C	89004	61035	57994	92048	1711	77543	34438	75411
D	35005	41080	23619	41159	718	41240	25248	28446
E	13586	24748	7364	18139	348	22313	15531	9285
F	3884	9113	1914	5742	118	8238	5661	2520
G	1043	2838	361	1689	81	2668	1667	555

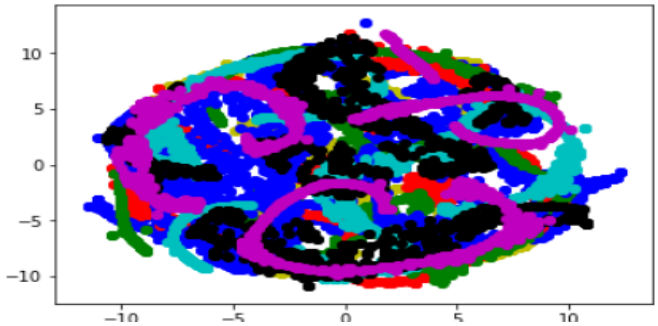
Here we Have the t-SNE graphs

t-sne code for each k-means cluster and the output is shown in the graph below. All the k-means are overlaid on each other below.

```
snegraph2 = TSNE(perplexity=200,n_iter=200)
```

```
T0 = snegraph2.fit_transform(sel_loan_data_0)
T1 = snegraph2.fit_transform(sel_loan_data_1)
T2 = snegraph2.fit_transform(sel_loan_data_2)
T3 = snegraph2.fit_transform(sel_loan_data_3)
T4 = snegraph2.fit_transform(sel_loan_data_4)
T5 = snegraph2.fit_transform(sel_loan_data_5)
T6 = snegraph2.fit_transform(sel_loan_data_6)
T7 = snegraph2.fit_transform(sel_loan_data_7)
```

```
plt.scatter(T0[:,0],T0[:,1], c='y')
plt.scatter(T1[:,0],T1[:,1], c='b')
plt.scatter(T2[:,0],T2[:,1], c='r')
plt.scatter(T3[:,0],T3[:,1], c='g')
plt.scatter(T4[:,0],T4[:,1], c='b')
plt.scatter(T5[:,0],T5[:,1], c='c')
plt.scatter(T6[:,0],T6[:,1], c='k')
plt.scatter(T7[:,0],T7[:,1], c='m')
plt.show()
```



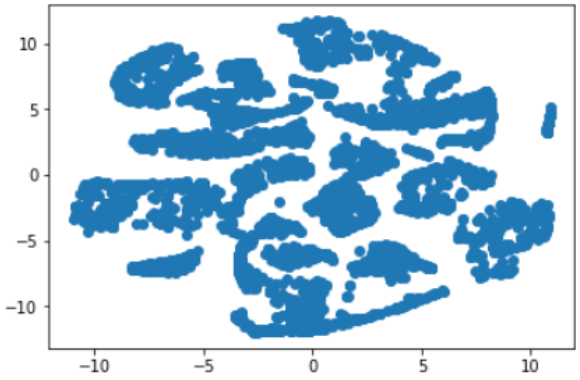
Below is t-sne for 0-cluster.

```
sel_loan_data = feature_df.sample(n=6000)
```

```
snegraph = TSNE(perplexity=100,n_iter=500)
```

```
Y = snegraph.fit_transform(sel_loan_data)
```

```
plt.scatter(Y[:,0],Y[:,1], cmap=plt.cm.Spectral)
plt.show()
```



2(c): Prediction

We have a file that is created for regression and in that we have run all 13 clusters. So a model for no cluster with below result:

```
Train
3.712511343697658
2.84995954278
23.633236194909287
Test
3.699412729925242
2.84114015302
23.562978838485247
```

Then for manual cluster being split into 4 different data frames then run through.

```
#Seperate into diff dataframes
manual_c1 = manual_df[manual_df['cluster']=='K1']
manual_c1.drop('cluster',axis=1,inplace=True)
manual_c2 = manual_df[manual_df['cluster']=='K2']
manual_c2.drop('cluster',axis=1,inplace=True)
manual_c3 = manual_df[manual_df['cluster']=='K3']
manual_c3.drop('cluster',axis=1,inplace=True)
manual_c4 = manual_df[manual_df['cluster']=='K4']
manual_c4.drop('cluster',axis=1,inplace=True)
```

Lastly we ran for K-Means Cluster

```
]: k_cluster = joblib.load(open('../kmeanCluster.pkl', 'rb'))
   #Create new column in load_df for the k-means clustering
   k_mean_feature_df = feature_df
   k_mean_feature_df['kMean'] = k_cluster.labels_
```

```
]: kmean_0 = k_mean_feature_df[k_mean_feature_df['kMean']==0]
   kmean_0.drop('kMean', axis=1, inplace=True)
   kmean_1 = k_mean_feature_df[k_mean_feature_df['kMean']==1]
   kmean_1.drop('kMean', axis=1, inplace=True)
   kmean_2 = k_mean_feature_df[k_mean_feature_df['kMean']==2]
   kmean_2.drop('kMean', axis=1, inplace=True)
   kmean_3 = k_mean_feature_df[k_mean_feature_df['kMean']==3]
   kmean_3.drop('kMean', axis=1, inplace=True)
   kmean_4 = k_mean_feature_df[k_mean_feature_df['kMean']==4]
   kmean_4.drop('kMean', axis=1, inplace=True)
   kmean_5 = k_mean_feature_df[k_mean_feature_df['kMean']==5]
   kmean_5.drop('kMean', axis=1, inplace=True)
   kmean_6 = k_mean_feature_df[k_mean_feature_df['kMean']==6]
   kmean_6.drop('kMean', axis=1, inplace=True)
   kmean_7 = k_mean_feature_df[k_mean_feature_df['kMean']==7]
   kmean_7.drop('kMean', axis=1, inplace=True)
```

Regression

The below code has been run for each cluster.

```
X_train, X_test, y_train, y_test = train_test_split(zero_cluster_df.ix[:, zero_cluster_df.columns != 'int_rate'],
                                                    zero_cluster_df['int_rate'],
                                                    test_size=0.30)
lm=linear_model.LinearRegression()
lm.fit(X_train,y_train)
y_train_predicted = lm.predict(X_train)
lm.fit(X_test,y_test)
y_test_predicted = lm.predict(X_test)
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, and number of units
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_train, y_train_predicted))
mae_train = mean_absolute_error(y_train, y_train_predicted)
mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
print("Train")
print(rms)
print(mae_train)
print(mape_train)
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, and number of units
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_test, y_test_predicted))
mae_test = mean_absolute_error(y_test, y_test_predicted)
mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100
print("Test")
print(rms)
print(mae_test)
print(mape_test)
```

Neural Network

```
# Creating the model
nn = MLPRegressor(hidden_layer_sizes=(50,25,5),max_iter=15,verbose=False, activation='relu',solver='adam')
nn.fit(X=X_train.values,y= y_train)
```

```
# Accuracy of the model created
accuracy = nn.score(X_train, y_train)
print(accuracy)
```

```
0.129757779772
```

```
y_test_predicted = nn.predict(X_test)
y_train_predicted= nn.predict(X_train)
```

```
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, and numbe
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_train, y_train_predicted))
mae_train = mean_absolute_error(y_train, y_train_predicted)
mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
print("-----Train-----")
print('RMS: ',rms)
print('MAE: ',mae_train)
print('MAPE: ',mape_train)
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, and numbe
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_test, y_test_predicted))
mae_test = mean_absolute_error(y_test, y_test_predicted)
mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100
print("-----Test-----")
print('RMS: ',rms)
print('MAE: ',mae_test)
print('MAPE: ',mape_test)
```

KNN

```
# Creating the model
clf = neighbors.KNeighborsRegressor()
clf.fit(X_train, y_train)
```

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform')
```

```
# Accuracy of the model created
accuracy = clf.score(X_train, y_train)
print(accuracy)

y_test_predicted = clf.predict(X_test)
y_train_predicted= clf.predict(X_train)

#finished square footage of the property, square footage of lot, tax amounts
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_train, y_train_predicted))
mae_train = mean_absolute_error(y_train, y_train_predicted)
mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
print("-----Train-----")
print('RMS: ',rms)
print('MAE: ',mae_train)
print('MAPE: ',mape_train)
#finished square footage of the property, square footage of lot, tax amounts
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_test, y_test_predicted))
mae_test = mean_absolute_error(y_test, y_test_predicted)
mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100
print("-----Test-----")
print('RMS: ',rms)
print('MAE: ',mae_test)
print('MAPE: ',mape_test)
```

Random Forest

```
rf = RandomForestRegressor(n_estimators=60)
rf.fit(X_train, y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=60, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=False)
```

```
# Accuracy of the model created
accuracy = rf.score(X_train, y_train)
print(accuracy)
```

```
0.91570957196
```

```
y_test_predicted = rf.predict(X_test)
y_train_predicted= rf.predict(X_train)
```

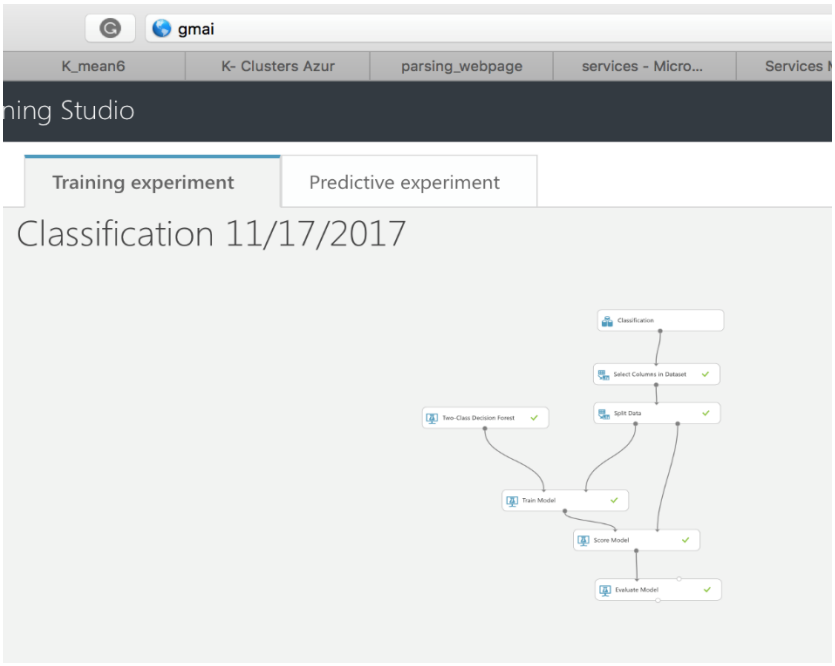
```
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, a
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_train, y_train_predicted))
mae_train = mean_absolute_error(y_train, y_train_predicted)
mape_train = np.mean(np.abs((y_train - y_train_predicted) / y_train)) * 100
print("-----Train-----")
print('RMS: ',rms)
print('MAE: ',mae_train)
print('MAPE: ',mape_train)
#finished square footage of the property, square footage of lot, tax amounts, number of bedrooms, a
#MAE, RMS, MAPE
rms = sqrt(mean_squared_error(y_test, y_test_predicted))
mae_test = mean_absolute_error(y_test, y_test_predicted)
mape_test = np.mean(np.abs((y_test - y_test_predicted) / y_test)) * 100
print("-----Test-----")
print('RMS: ',rms)
print('MAE: ',mae_test)
print('MAPE: ',mape_test)
```

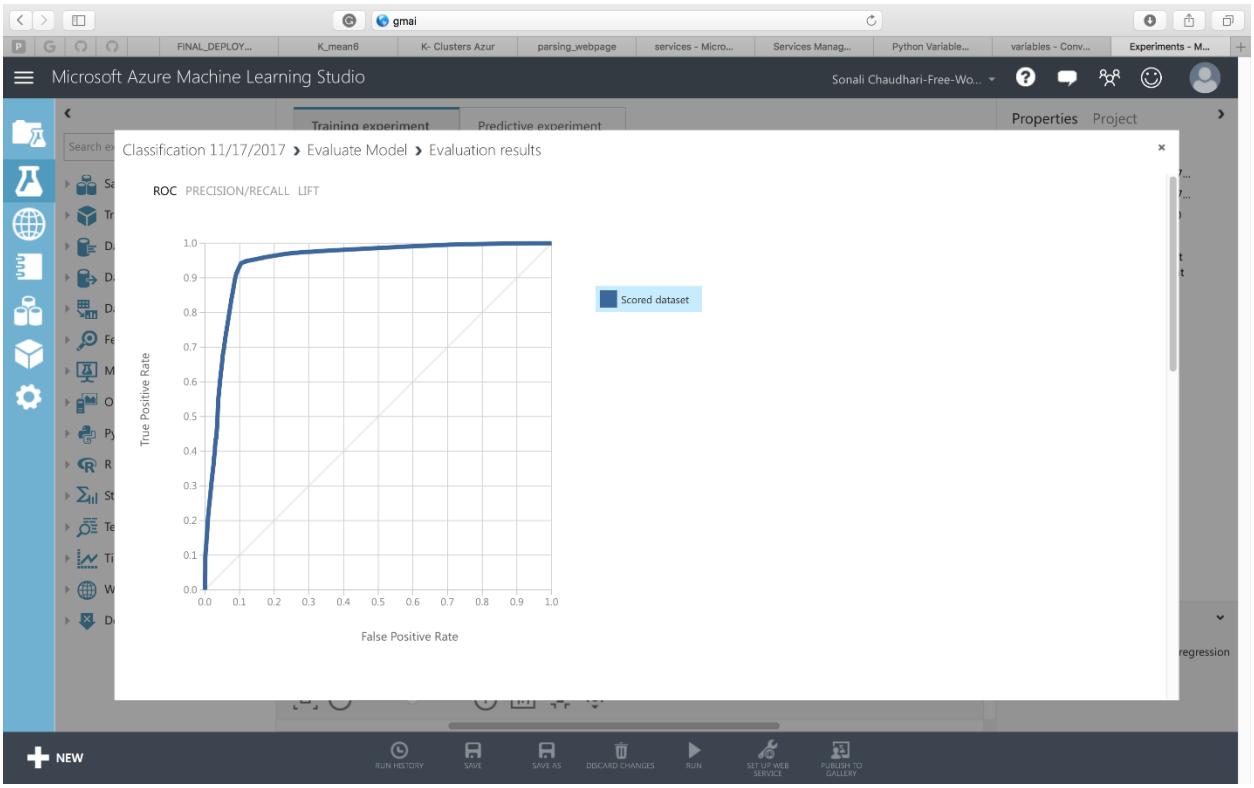
We have Created A Results_Cluster excel file which has all the training and testing outputs for prediction.

2(d): Deployment

Microsoft Azur

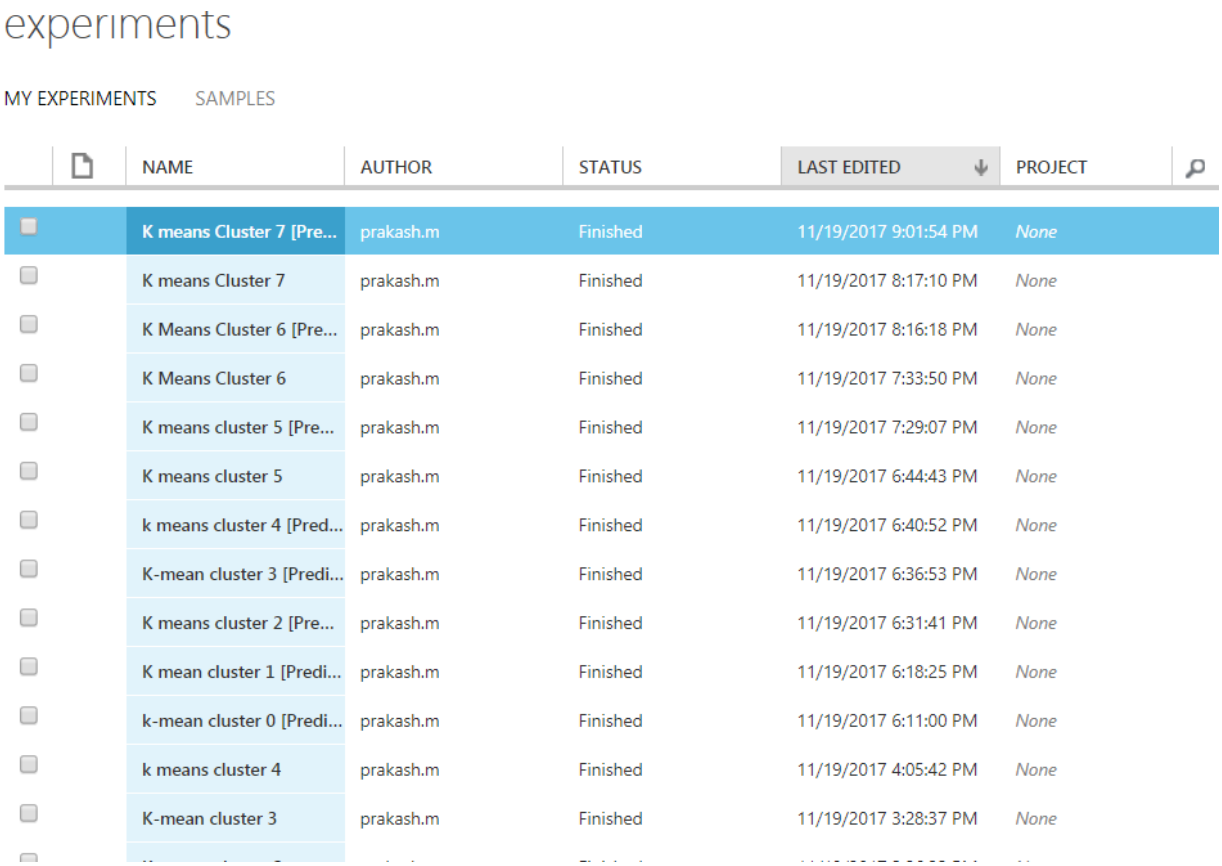
We Have Done RANDOM FOREST for classification





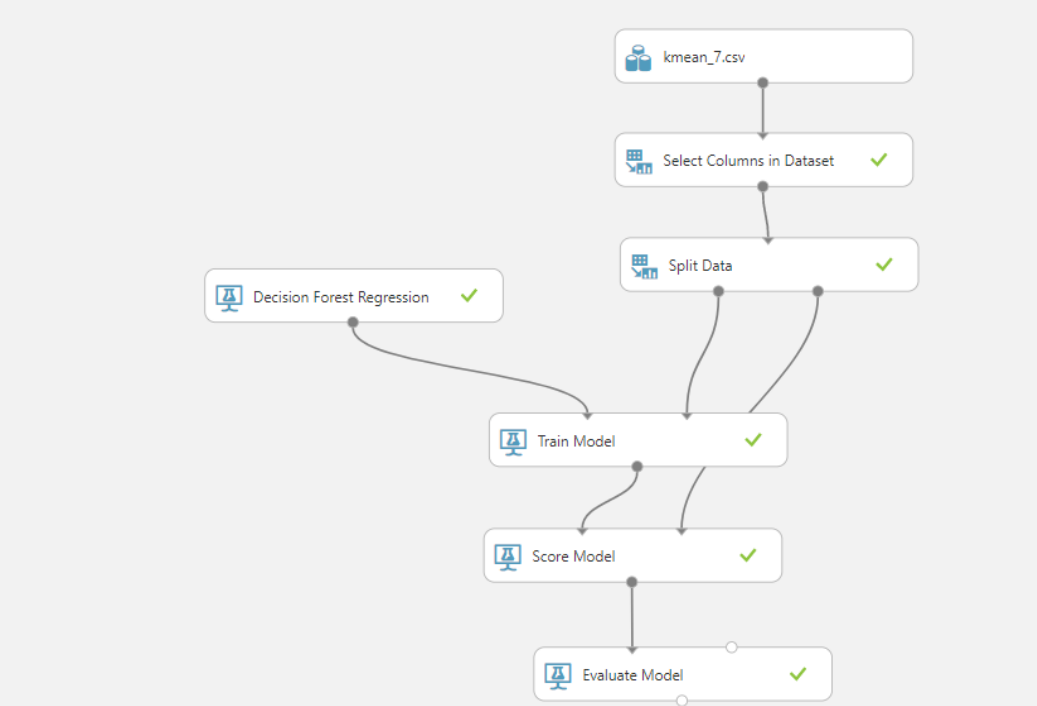
We have selected RANDOM FOREST to deploy as the best model for all 13 clusters (1 0-cluster, 4 manual clusters, and 8 K-means cluster)

Experiments were created using a csv for each type of cluster.

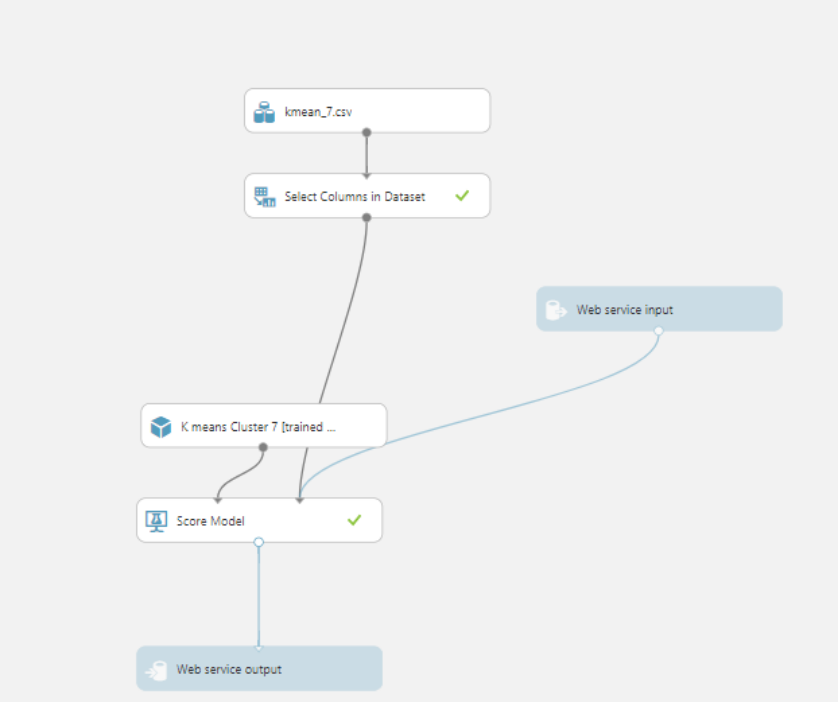


NAME	AUTHOR	STATUS	LAST EDITED	PROJECT
K means Cluster 7 [Pre...	prakash.m	Finished	11/19/2017 9:01:54 PM	None
K means Cluster 7	prakash.m	Finished	11/19/2017 8:17:10 PM	None
K Means Cluster 6 [Pre...	prakash.m	Finished	11/19/2017 8:16:18 PM	None
K Means Cluster 6	prakash.m	Finished	11/19/2017 7:33:50 PM	None
K means cluster 5 [Pre...	prakash.m	Finished	11/19/2017 7:29:07 PM	None
K means cluster 5	prakash.m	Finished	11/19/2017 6:44:43 PM	None
k means cluster 4 [Pred...	prakash.m	Finished	11/19/2017 6:40:52 PM	None
K-mean cluster 3 [Predi...	prakash.m	Finished	11/19/2017 6:36:53 PM	None
K means cluster 2 [Pre...	prakash.m	Finished	11/19/2017 6:31:41 PM	None
K mean cluster 1 [Predi...	prakash.m	Finished	11/19/2017 6:18:25 PM	None
k-mean cluster 0 [Predi...	prakash.m	Finished	11/19/2017 6:11:00 PM	None
k means cluster 4	prakash.m	Finished	11/19/2017 4:05:42 PM	None
K-mean cluster 3	prakash.m	Finished	11/19/2017 3:28:37 PM	None

Below you will be able to see how we modeled a sample prediction model using cluster csv. All clusters have same model but diff csv.



We then setup web service. In this step we change the "selected column" and take out the interest rate as we are calculating that, and any other columns that we do not want.



After that we deploy and get API and we go to "Test" and go to config to get python 3+ code. We have 13 AIP and below we can see the code for python.

```
import urllib.request
import json

data = {
        "Inputs": {
                "input1":
                [
                    {
                            'debt_settlement_flag': "N",
                            'application_type': "Individual",
                            'fico_range_low': "735",
                            'fico_range_high': "739",
                            'emp_length': "10",
                            'dti': "27.65",
                            'annual_inc': "24000",
                            'grade': "B",
                            'sub_grade': "B2",
                            'loan_amnt': "5000",
                            'issue_d': "2011-12-01T00:00:00Z",
                            'purpose': "credit_card",
                            'State': "AZ",
                            'home_ownership': "RENT",
                            'zip_code': "860",
                            'policy_code': "1",
                            'term': "36",
                            'Year': "2011",
                            'Month': "12",
                            'fico': "737",
                            'approval': "1",
                    }
                ],
        },
    "GlobalParameters":  {
    }
}

body = str.encode(json.dumps(data))

url = 'https://ussouthcentral.services.azureml.net/workspaces/c2f8dc5126ac4024b08690eda8426a56/services/11904d20378745bf9cad7d40121722c3/execut
e?api-version=2.0&format=swagger'
api_key = 'abc123' # Replace this with the API key for the web service
headers = {'Content-Type':'application/json', 'Authorization':('Bearer '+ api_key)}

req = urllib.request.Request(url, body, headers)

try:
    response = urllib.request.urlopen(req)

    result = response.read()
    print(result)
except urllib.error.HTTPError as error:
    print("The request failed with status code: " + str(error.code))

    # Print the headers - they include the request ID and the timestamp, which are useful for debugging the failure
```

We then wrote a jupyter script to use these AIP and payload to first use classification to determine if to give loan or not. Then we run for each of the 3 clusters and pick the best interest rate.

Deployment

- There are bunch of input values a user has to give like
  - Loan Amount
  - Income
  - Debt to Income Ratio
  - Zipcode
  - State
  - Year
  - Month
  - Length of Employment

- No Cluster: First we pass these parameters and pass to the No Cluster deployment code inputs and get the predicted interest rate after parsing the result and store it in ‘rate1’
- Manual Cluster: Then we check for the fico score the user entered and get the cluster which it belongs to. The we pass those inputs to those cluster deployment code and get the interest rate predicted and store it on ‘rate2’.
- K-Means: We pass the input parameters to all the 8 clusters we have and get the interest rates predicted for each cluster. We get the max of all the resulted predicted interest rates for K-Means clusters and store it in ‘rate3’.
- Finally we have 3 interest rates predicted for our 3 clustering methods.
- We get the best interest rate by taking the max of the rate1, rate2, rate3.

```
In [21]: rate = max(rate1,rate2,rate3)

In [23]: print('The best interest rate is: ',rate,'%')
         The best interest rate is:  30.79 %
```

-

Analysis

While looking at the data we realized that there were 157 columns. We have removed everything about 70% missing. When analyzing this data, we realized that there in decline data, there was a lot of risk score data missing. There was no point of filling in the data as it has no basis of what to fill it o since there was only 9 columns and second we will change the data a lot if we do so. We therefore took a look at the data and removed 1/3rd of the decline data on that basis.
Also in loan data we realized a fico score below 660 is not correct as it was mentioned in the lending club site. A person must have at least that to get a loan. Also, anything below 330 in decline for risk score was removed, since someone cannot have that as a possible fico score.
We have also looked at the relationships between the columns to input the data using knn like annual income and interest rate. Also, there were 2 states when looking into the site that would have to be removed since lending club is illegal there.

Looking at accuracy between classification and prediction, we can see that classification is much better. This would be mostly be because there is a strong distinction as to what kind of data combination will cause someone to get interest rate or not. Also lending club probably only really uses these 9 columns to determine whether to give a loan or not. However, prediction gives an interest rate and we are only considering a certain number of fields. In real life though lending club would be taking into consideration way more than what we did.