

LOAN LENDING CLUB

Report by: Mengqi Zhao (Team 6)

Part1:

CREATE USER ID AND PASSWORD TO DOWNLOAD LOAN DATA FILES

Given user name and password by the user, we will be programmatically generating the URL to download the various data file present on the Loan Lending Club Website. We also do have sample file present on their website which does not require any user login but it consist of only sample data with less number of columns.

```
In [1]: import luigi
import requests
import sys
import logging
import pandas as pd
import numpy as np
import glob
import os
from bs4 import BeautifulSoup
from zipfile import ZipFile
from io import BytesIO
from luigi.parameter import MissingParameterException
```

```
In [5]: def extractZip(yearwisedata,path):
    r = requests.get(yearwisedata)
    z = ZipFile(BytesIO(r.content))
    z.extractall(path)

def output(self):
    return luigi.LocalTarget('rejectdictionary.csv')

url = 'https://www.lendingclub.com/account/login.action?'
postUrl = 'https://www.lendingclub.com/info/download-data.action'
payload = {'login_email': 'zhaomengqi8000@gmail.com', 'login_password': 'billy8213198'}
with requests.Session() as s:
    loginRequest = s.post(url, data=payload)
    finalUrl = s.get(postUrl)
    linkhtml = finalUrl.text
    soup = BeautifulSoup(linkhtml, "html.parser")
    zipList = soup.findAll('div', {"id": 'rejectedLoanStatsFileNamesJS'})
    Borrowerdata = []
    rejectLoandata = []
    rejectloanyearlist = []
    for div in zipList:
        for d in div:
            link = d.split('|')
            Borrowerdata.extend(link)
    for data in Borrowerdata:
        if data != '':
            rejectLoandata.append('https://resources.lendingclub.com/' + data)
    year_options = soup.findAll('select', {"id": 'rejectStatsDropdown'})[0].findAll("option")
    for year in year_options:
        rejectloanyearlist.append(year.text)
    dictionary = dict(zip(rejectloanyearlist, rejectLoandata))
    # print(dictionary)
    for year in dictionary:
        url = dictionary[year]
        yearpath = str(os.getcwd()) + "/RejectData/" + year
        extractZip(url, yearpath)
    yeardict=pd.DataFrame(list(dictionary.items()),columns=['year','Link'])
    yeardict.to_csv('rejectdictionary.csv', mode='a', encoding='utf-8', index=False)
```

```

1 year,Link
2 2007 - 2012,https://resources.lendingclub.com/RejectStatsA.csv.zip
3 2013 - 2014,https://resources.lendingclub.com/RejectStatsB.csv.zip
4 2015,https://resources.lendingclub.com/RejectStatsD.csv.zip
5 2016 Q1,https://resources.lendingclub.com/RejectStats_2016Q1.csv.zip
6 2016 Q2,https://resources.lendingclub.com/RejectStats_2016Q2.csv.zip
7 2016 Q3,https://resources.lendingclub.com/RejectStats_2016Q3.csv.zip
8 2016 Q4,https://resources.lendingclub.com/RejectStats_2016Q4.csv.zip
9 2017 Q1,https://resources.lendingclub.com/RejectStats_2017Q1.csv.zip
10 2017 Q2,https://resources.lendingclub.com/RejectStats_2017Q2.csv.zip
11 2017 Q3,https://resources.lendingclub.com/RejectStats_2017Q3.csv.zip
12 year,Link
13 2007 - 2012,https://resources.lendingclub.com/RejectStatsA.csv.zip
14 2013 - 2014,https://resources.lendingclub.com/RejectStatsB.csv.zip
15 2015,https://resources.lendingclub.com/RejectStatsD.csv.zip
16 2016 Q1,https://resources.lendingclub.com/RejectStats_2016Q1.csv.zip
17 2016 Q2,https://resources.lendingclub.com/RejectStats_2016Q2.csv.zip
18 2016 Q3,https://resources.lendingclub.com/RejectStats_2016Q3.csv.zip
19 2016 Q4,https://resources.lendingclub.com/RejectStats_2016Q4.csv.zip
20 2017 Q1,https://resources.lendingclub.com/RejectStats_2017Q1.csv.zip
21 2017 Q2,https://resources.lendingclub.com/RejectStats_2017Q2.csv.zip
22 2017 Q3,https://resources.lendingclub.com/RejectStats_2017Q3.csv.zip
23

```

```

In [3]: writeHeader2 = True
for year in dictionary:
    # for year in customyear:
        yearpath = str(os.getcwd()) + "\\\" + year
        for filename in os.listdir(yearpath):
            print(filename)
            newFile = "ModifiedData.csv"
            for f in glob.glob(yearpath + '\\\' + filename):
                datadf = pd.read_csv(f, skiprows=1, skipfooter=2, engine='python')
                with open(newFile, 'a', encoding='utf-8', newline="") as file:
                    for f in glob.glob(str(os.getcwd()) + '\\\' + newFile):
                        if writeHeader2 is True:
                            datadf.to_csv(f, mode='w', header=True, index=False, skipinitialspace=True)
                            writeHeader2 = False
                        else:
                            datadf.to_csv(f, mode='a', header=False, index=False, skipinitialspace=True)

```

```

LoanStats3a_securev1.csv
LoanStats3b_securev1.csv
LoanStats3c_securev1.csv
LoanStats3d_securev1.csv
LoanStats_securev1_2016Q1.csv
LoanStats_securev1_2016Q2.csv
LoanStats_securev1_2016Q3.csv
LoanStats_securev1_2016Q4.csv
LoanStats_securev1_2017Q1.csv
LoanStats_securev1_2017Q2.csv
LoanStats_securev1_2017Q3.csv

```

```
In [5]: def extractZip(yearwisedata,path):
    r = requests.get(yearwisedata)
    z = ZipFile(BytesIO(r.content))
    z.extractall(path)

url = 'https://www.lendingclub.com/account/login.action?'
postUrl = 'https://www.lendingclub.com/info/download-data.action'
payload = {'login_email': 'zhaomengqi8000@gmail.com', 'login_password': 'billy8213198'}
with requests.Session() as s:
    loginRequest = s.post(url, data=payload)
    finalUrl = s.get(postUrl)
    linkhtml = finalUrl.text
    soup = BeautifulSoup(linkhtml, "html.parser")
    ziplist = soup.find_all('div', {"id": 'rejectedLoanStatsFileNamesJS'})
    Borrowerdata = []
    rejectLoandata = []
    rejectloanyearlist = []
    for div in ziplist:
        for d in div:
            link = d.split('|')
            Borrowerdata.extend(link)
    for data in Borrowerdata:
        if data != '':
            rejectLoandata.append('https://resources.lendingclub.com/' + data)
    year_options = soup.findAll('select', {"id": 'rejectStatsDropdown'})[0].findAll("option")
    for year in year_options:
        rejectloanyearlist.append(year.text)
    dictionary = dict(zip(rejectloanyearlist, rejectLoandata))
    # print(dictionary)
    for year in dictionary:
        url = dictionary[year]
        yearpath = str(os.getcwd()) + "\\RejectData\\" + year
        extractZip(url, yearpath)
    yeardict=pd.DataFrame(list(dictionary.items()),columns=['year','Link'])
    yeardict.to_csv('rejectdictionary.csv', mode='a', encoding='utf-8', index=False)
```

```
In [40]: # consolidate date
writeHeader2 = True
for year in dictionary:
    yearpath = str(os.getcwd()) + "\\RejectData\\" + year

    for filename in os.listdir(yearpath):
        print(filename)
        newFile = "RejectModifiedData.csv"

        for f in glob.glob(yearpath + '\\' + filename):

            datadf = pd.read_csv(f, skiprows=1, skipfooter=2, engine='python')
            with open(newFile, 'a', encoding='utf-8', newline="")as file:
                for f in glob.glob(str(os.getcwd()) + '\\' + newFile):
                    datadf.to_csv('RejectModifiedData.csv', mode='a', encoding='utf-8')
```

RejectStatsA.csv
 RejectStatsB.csv
 RejectStatsD.csv
 RejectStats_2016Q1.csv
 RejectStats_2016Q2.csv
 RejectStats_2016Q3.csv
 RejectStats_2016Q4.csv
 RejectStats_2017Q1.csv
 RejectStats_2017Q2.csv
 RejectStats_2017Q3.csv

DATA CLEANING & HANDLING MISSING VALUE

Once we have all the data downloaded as a single csv file, we will load the resultant dataset in pandas data frame for data cleaning and handling missing value.

```
: loan_df = pd.read_csv('ModifiedData.csv', low_memory=False, encoding='ISO-8859-1', skipinitialspace=True)
loan_df.shape
: (1321848, 115)

: loan_df.columns
: Index(['id', 'member_id', 'loan_amnt', 'funded_amnt', 'funded_amnt_inv',
       'term', 'int_rate', 'installment', 'grade', 'sub_grade',
       ...
       'num_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'pct_tl_nvr_dlq',
       'percent_bc_gt_75', 'pub_rec_bankruptcies', 'tax_liens',
       'tot_hi_cred_lim', 'total_bal_ex_mort', 'total_bc_limit',
       'total_il_high_credit_limit'],
      dtype='object', length=115)
```

```
Out[6]: Amount Requested      0
Application Date      0
Loan Title            0
Risk_Score            0
Debt-To-Income Ratio  0
State                  0
Employment Length     0
Policy Code           0
dtype: int64
```

```
In [8]: reject_df.shape
Out[8]: (11079386, 8)
```

We will now remove all the unused columns which are not useful for the loan data analysis or consist of data which is insignificant. Also, we drop all the column having Null value more than 80% in our data set.

```
print ('Removing unused columns')
loan_df = loan_df[loan_df.id != 'Loans that do not meet the credit policy']
loan_df.drop(['id', 'member_id', 'emp_title','pymnt_plan','url','desc','title' ],axis=1, inplace=True)
column_naper_dict = {}
print ('Removing columns with nan% > 80%')
for column in loan_df:
    if loan_df[column].isnull().sum()>0:
        column_naper_dict[column] = loan_df[column].isnull().sum() / 1321847
        if column_naper_dict[column] > 0.80:
            loan_df.drop(column, axis=1,inplace=True)
```

```
Removing unused columns
Removing columns with nan% > 80%
```

```
#Handling missing data
print ('Handling missing data')
loan_df.term = pd.to_numeric(loan_df.term.str[:3])
loan_df["int_rate"] = pd.Series(loan_df.int_rate).str.replace('%','')
loan_df["revol_util"] = pd.Series(loan_df.revol_util).str.replace('%','')
loan_df.replace('n/a', np.nan,inplace=True)
loan_df.emp_length.fillna(value=0,inplace=True)
loan_df['emp_length'].replace(to_replace='[^0-9]+', value='', inplace=True, regex=True)
loan_df['emp_length'] = loan_df['emp_length'].astype(int)
loan_df["annual_inc"].fillna(loan_df["annual_inc"].median(), inplace=True)
loan_df["issue_d"] = loan_df["issue_d"].str.split("-")
loan_df["issue_month"] = loan_df["issue_d"].str[0]
loan_df["issue_year"] = loan_df["issue_d"].str[1]
m = loan_df['mths_since_last_delinq'].max() #max is 188 so this will be our imputed value.
loan_df['mths_since_last_delinq'] = np.where(loan_df['mths_since_last_delinq'].isnull(), m, loan_df['mths_since_last_delinq'])
#Revol_util will involve a median value imputation.
loan_df['revol_util'] = loan_df['revol_util'].fillna(loan_df['revol_util'].median())
loan_df['tot_coll_amt'] = loan_df['tot_coll_amt'].fillna(loan_df['tot_coll_amt'].median())
#tot_cur_bal will be fixed in similar manner.
loan_df['tot_cur_bal'] = loan_df['tot_cur_bal'].fillna(loan_df['tot_cur_bal'].median())
#total_rev_hi_lim will also contain median imputation
loan_df['total_rev_hi_lim'] = loan_df['total_rev_hi_lim'].fillna(loan_df['total_rev_hi_lim'].median())
loan_df['earliest_cr_line'] = loan_df['earliest_cr_line'].fillna('Unknown')
loan_df['last_pymnt_d'] = loan_df['last_pymnt_d'].fillna('Unknown')
loan_df['next_pymnt_d'] = loan_df['next_pymnt_d'].fillna('Unknown')
loan_df['last_credit_pull_d'] = loan_df['last_credit_pull_d'].fillna('Unknown')
loan_df['last_fico_range'] = loan_df.last_fico_range_low.astype('str') + '-' + loan_df.last_fico_range_high.astype('str')
loan_df['last_meanfico'] = (loan_df.last_fico_range_low + loan_df.last_fico_range_high)/2
loan_df = loan_df.fillna(0)
loan_df = changedatatype(loan_df)
print ('Creating Cleaned CSV file')
loan_df.to_csv('Final.csv', header=True,index=False,skipinitialspace=True)
```

Changing the data types of the columns:

```
def changedatatype(df):
    #Change the data types for all column
    df[['loan_amnt','funded_amnt','funded_amnt_inv','term','int_rate','installment','grade','sub_grade','emp_length','home_ownership','...','pub_rec_bankruptc']] = df[['loan_amnt','funded_amnt','funded_amnt_inv','term','int_rate','installment','grade','sub_grade','emp_length','home_ownership','...','pub_rec_bankruptc']].astype('float64')
    df[['mths_since_last_delinq','open_acc','pub_rec','revol_bal','total_acc','last_meanfico']] = df[['mths_since_last_delinq','open_acc','pub_rec','revol_bal','total_acc','last_meanfico']].astype('int64')
    return df
```

Cleaned Loan Data Set:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership	...	pub_rec_bankruptc
0	10400	10400	10400	36	6.99	321.08	A	A3	8	MORTGAGE	...	0.0
1	15000	15000	15000	60	12.39	336.64	C	C1	10	RENT	...	0.0
2	21425	21425	21425	60	15.59	516.36	D	D1	6	RENT	...	0.0
3	7650	7650	7650	36	13.66	260.20	C	C3	1	RENT	...	0.0
4	12800	12800	12800	60	17.14	319.08	D	D4	10	MORTGAGE	...	0.0

5 rows × 108 columns

Similarly, all the operation was performed on the Rejected loan data sets:

```

def changedatatype(df):
    #Change the data types for all column
    df[['Amount Requested','Risk_Score']] = df[['Amount Requested','Risk_Score']].astype('int64')
    df[['Debt-To-Income Ratio']] = df[['Debt-To-Income Ratio']].astype('float64')
    return df

def handleMissingData(reject_df):
    print ('Removing unused columns')

    reject_df.drop(['Zip Code'],axis=1, inplace=True)
    column_naper_dict = {}
    print ('Removing columns with nan% > 80%')
    for column in reject_df:
        if reject_df[column].isnull().sum()>0:
            column_naper_dict[column] = reject_df[column].isnull().sum()/ 11079372
            if column_naper_dict[column] > 0.80:
                reject_df.drop(column, axis=1,inplace='True')

    #Handling missing data
    print ('Handling missing data')
    reject_df["Debt-To-Income Ratio"]= pd.Series(reject_df['Debt-To-Income Ratio']).str.replace('%','')
    reject_df.replace('n/a', np.nan,inplace=True)
    reject_df['Employment Length'].fillna(value=0,inplace=True)
    reject_df.replace(to_replace='[^0-9]+', value='', inplace=True, regex=True)
    reject_df['Employment Length'] = reject_df['Employment Length'].astype(int)
    reject_df["Risk_Score"].fillna(0, inplace=True)
    reject_df['Loan Title'] = reject_df['Loan Title'].fillna('other')
    reject_df['State'] = reject_df['State'].fillna('NA')
    reject_df = changedatatype(reject_df)
    print ('Creating Cleaned CSV file')
    reject_df.to_csv('RejectLoan.csv', header=True,index=False,skipinitialspace=True)
    return reject_df

```

```

: file='RejectModifiedData.csv'
reject_df = pd.read_csv(file,low_memory=False,encoding='ISO-8859-1',skipinitialspace=True)
reject_df=handleMissingData(reject_df)
reject_df.head()

```

Removing unused columns
 Removing columns with nan% > 80%
 Handling missing data
 Creating Cleaned CSV file

	Amount Requested	Application Date	Loan Title	Risk_Score	Debt-To-Income Ratio	State	Employment Length	Policy Code
0	30000	2013-01-01	debt_consolidation	754	32.01	MN	1	0
1	15000	2013-01-01	medical	728	27.01	CA	1	0
2	2500	2013-01-01	other	723	0.92	PA	7	0
3	4000	2013-01-01	car	563	24.92	SC	1	0
4	22000	2013-01-01	debt_consolidation	0	26.91	OK	1	0

FEATURE ENGINEERING

```
#Feature Engineering
import warnings
from sklearn.linear_model import (LinearRegression, Ridge,
                                    Lasso, RandomLasso)
from sklearn.feature_selection import RFE, f_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

```
#Loading dataset for feature engineering
print("Training Data")
loanfeature_df=pd.read_csv('Final.csv',low_memory=False,encoding='ISO-8859-1',skipinitialspace=True)
```

Training Data

```
#Creating dummy Variables
def createDummies(df):
    dummies1 = pd.get_dummies(df['purpose']).rename(columns=lambda x: 'purpose' + str(x))
    df=pd.concat([df, dummies1], axis=1)
    dummies2 = pd.get_dummies(df['application_type']).rename(columns=lambda x: 'application_type' + str(x))
    df=pd.concat([df, dummies2], axis=1)
    return df
```

We have changed some of the category columns into numeric.

```
#Dropping interest rate (target to predict)
Y =loanfeature_df.int_rate
loanfeature_df.drop('int_rate',axis=1,inplace=True)
#Creating Categorical Variables
home_positive = ['OWN', 'MORTGAGE']
home_negative = ['RENT','NONE','OTHER','ANY']
# filter out any word that is not within home_positive & home_negative
loanfeature_df = loanfeature_df[loanfeature_df['home_ownership'].isin(home_positive + home_negative)].copy()
loanfeature_df['home_ownership_category']=loanfeature_df['home_ownership'].isin(home_positive).astype(int)
# Make "verified" and "Source Verified" in the "verification_status" column as 1 and non-verified as 0
verification_positive = ['Verified', 'Source Verified']
verification_negative = ['Not Verified']
loanfeature_df = loanfeature_df[loanfeature_df['verification_status'].isin(verification_positive + verification_negative)]
loanfeature_df['verification_status_category']=loanfeature_df['verification_status'].isin(verification_positive).astype(int)
```

Our analysis suggests that FICO score is not the only parameter predicting Interest Rate and using statistical methods such as multivariate linear regression we show that there is obvious relationship between Interest Rate and the following parameters: Loan length, amount of money funded by investors, debt to income ratio and inquiries in the last 6 months

```
print ('Removing unused column for feature engineering')
cols_to_keep=['loan_amnt','term','emp_length','home_ownership_category','annual_inc',
              'verification_status_category','purpose','addr_state','dti','delinq_2yrs',
              'last_meanfico','inq_last_6mths','open_acc','revol_bal','revol_util','total_acc',
              'mths_since_last_major_derog','funded_amnt_inv','installment','application_type','pub_rec']
loanfeature_df = loanfeature_df[cols_to_keep]
loanfeature_df = createDummies(loanfeature_df)
X = loanfeature_df._get_numeric_data()
```

Removing unused column for feature engineering

```
def rank_to_dict(ranks, names, order=1):
    minmax = MinMaxScaler()
    ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
    ranks = map(lambda x: round(x, 2), ranks)
    return dict(zip(names, ranks ))
```

```

#Comparing models for feature selection
names = ["%s" % i for i in X]
ranks = {}

lr = LinearRegression(normalize=True)
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    lr.fit(X, Y)
    ranks["Linear reg"] = rank_to_dict((lr.coef_), names)

ridge = Ridge(alpha=7)
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    ridge.fit(X, Y)
    ranks["Ridge"] = rank_to_dict((ridge.coef_), names)

lasso = Lasso(alpha=.05)
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    lasso.fit(X, Y)
    ranks["Lasso"] = rank_to_dict(np.abs(lasso.coef_), names)

rlasso = RandomizedLasso(alpha=0.00)
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    rlasso.fit(X, Y)
    ranks["Stability"] = rank_to_dict((rlasso.scores_), names)

rf = RandomForestRegressor()
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    rf.fit(X, Y)
    ranks["RF"] = rank_to_dict(rf.feature_importances_, names)

# stop the search when 5 features are left (they will get equal scores)
rfe = RFE(lr, n_features_to_select=15)
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=DeprecationWarning)
    rfe.fit(X, Y)
    ranks["RFE"] = rank_to_dict(rfe.ranking_, X.columns, order=-1)

f, pval = f_regression(X, Y, center=True)
ranks["Corr."] = rank_to_dict(f, names)

r = {}
for name in names:
    r[name] = round(np.mean([ranks[method][name] for method in ranks.keys()]), 2)
methods = sorted(ranks.keys())
ranks["Mean"] = r
methods.append("Mean")

#     f_rank = pd.DataFrame()
print ("%t%" * "\t").join(methods)
temp= "\t".join(methods)
f=open("testing.txt", 'w')
f.write(temp)
f.write("\n")
for name in names:
    temp=name+"\t"+ "\t".join(map(str,
                                    [ranks[method][name] for method in methods]))
    f.write(temp)
    f.write("\n")
f.close()

```

Result :

The analyzed data contains information on the following parameters:

- Amount of requested money in dollars (Amount.Requested)
- Amount of money that was loaned to the individual(Amount.Funded.By.Investors)
- Length of time of the loan in months (Loan.Length)
- Purpose of a loan as stated by the applicant (Loan.Purpose)
- The percentage of customer gross income that goes toward paying debt (Debt.To.Income.Ratio)
- The abbreviation for the U.S. state of residence of (State)
- A variable indicating whether the applicant owns, rents or has a mortgage on his home (Home.ownership)
- The monthly income of the applicant in dollars (Monthly.income)
- A measure of the creditworthiness of the applicant, FICO Score (FICO.Range)
- The number of open lines of credit the applicant had at the time of application (Open.CREDIT.Lines)
- The total amount outstanding all lines of credit (Revolving.CREDIT.Balance)
- The number of authorized queries about the applicant's creditworthiness in the 6 months before the credit was issued (Inquiries.in.the.Last.6.Months)
- Length of time employed at current job (Employment.Length)

	Corr.	Lasso	Linear r	RF	RFE	Ridge	Stability	Mean
term	1	0.63	0.02	0.59	0.55	0.53	1	0.62
last_meanfico	0.53	0.01	0.02	0.38	0.32	0.42	1	0.38
revol_util	0.31	0.03	0.02	0.14	0.41	0.42	1	0.33
verification_status_category	0.26	0.84	0.02	0.04	0.77	0.61	1	0.51
inq_last_6mths	0.24	0.75	0.02	0.1	0.73	0.57	1	0.49
purposecredit_card	0.15	1	0.04	0.05	0.91	0	1	0.45
installment	0.11	0.06	0.02	1	0.23	0.43	1	0.41
loan_amnt	0.1	0	0.02	0.48	0.18	0.42	1	0.31
funded_amnt_inv	0.1	0	0.02	0.28	0.09	0.42	0.98	0.27
purposedebt_consolidation	0.05	0.01	0.04	0.01	1	0.22	0.55	0.27
annual_inc	0.03	0	0.02	0.14	0	0.42	1	0.23
purposeother	0.03	0.56	0.04	0.01	1	0.58	0.99	0.46
home_ownership_category	0.02	0.21	0.02	0.01	0.68	0.33	1	0.32

Correlation analysis showed high negative correlation between Interest rate and FICO score and moderate positive correlation between interest rate and loan length and amount of money funded by investors.

There also is slight positive correlation between interest rate and inquiries in the last six months and quantity of open credit lines.

Rejected Loan Data Files:

As we don't have target variable to predict, we choose to see the correlation between the variable.

correlation_df		
Amount Requested	Amount Requested	0.000000
	Risk_Score	0.000000
	Debt-To-Income Ratio	0.000000
	Employment Length	0.000000
	Policy Code	0.000000
Risk_Score	Amount Requested	-0.000399
	Risk_Score	0.000000
	Debt-To-Income Ratio	0.000000
	Employment Length	0.000000
	Policy Code	0.000000
Debt-To-Income Ratio	Amount Requested	0.000243
	Risk_Score	0.001839
	Debt-To-Income Ratio	0.000000
	Employment Length	0.000000
	Policy Code	0.000000
Employment Length	Amount Requested	0.005622
	Risk_Score	-0.039622
	Debt-To-Income Ratio	-0.000886
	Employment Length	0.000000
	Policy Code	0.000000
Policy Code	Amount Requested	0.042160
	Risk_Score	-0.026878
	Debt-To-Income Ratio	-0.000579
	Employment Length	0.063920
	Policy Code	0.000000
dtype: float64		

Also, these were the entries which were provided by the requester while requesting a loan from Lending Club.

- **Amount Requested**
- **Fico_Score**
- **Annual Income**
- **Employment Length**

PIPELINING PROCESS USING LUIGI

Luigi is a Python-based framework for expressing data pipelines. Everything in Luigi is in Python.

Luigi performs following functions in a pipeline for Loan data which is approved:

1. getWebUrls-Fetching urls for file download
2. getData-Merging the data from all downloaded file into a single file
3. handleMissingData-Managed missing data in the file
4. processData-Intermediate step for featureselection
5. featureSelection –Selects relevant features by applying several feature selection techniques.

The screenshot shows the Luigi web interface with the following sections:

- TASK FAMILIES:** A sidebar listing task families: featureSelection, getData, getWebUrls, handleMissingData, and processData.
- PENDING TASKS:** 2 tasks (orange icon).
- RUNNING TASKS:** 1 task (blue play icon).
- BATCH RUNNING TASKS:** 0 tasks (purple play icon).
- DONE TASKS:** 2 tasks (green checkmark icon).
- FAILED TASKS:** 0 tasks (red X icon).
- UPSTREAM FAILURE:** 0 tasks (pink exclamation mark icon).
- DISABLED TASKS:** 0 tasks (grey minus icon).
- UPSTREAM DISABLED:** 0 tasks (grey warning icon).

Show 10 entries

Name	Details	Priority	Time	Actions
RUNNING handleMissingData	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:45:19 PM 2 minutes	[Edit, Delete]
DONE getData	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:45:19 PM	[Edit, Delete]
DONE getWebUrls	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:47:51 PM	[Edit, Delete]
PENDING featureSelection	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:48:11 PM	[Edit, Delete]
PENDING processData	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:48:03 PM	[Edit, Delete]

Pipeline is completed when all the tasks into it are completed i.e. green status.



Luigi Task Status

PENDING TASKS		RUNNING TASKS		BATCH RUNNING TASKS		DONE TASKS	
	0		0		0		5
FAILED TASKS		UPSTREAM FAILURE		DISABLED TASKS		UPSTREAM DISABLED	
	0		0		0		0

Show 10 entries

Name	Details	Priority	Time	Actions
DONE	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:44:21 PM	
DONE	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:51:56 PM	
DONE	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 4:16:52 PM	
DONE	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:37:51 PM	
DONE	loginpassword-[REDACTED] loginemail-[REDACTED]	0	11/17/2017 3:58:51 PM	

Similarly, we implemented Luigi pipeline for rejected data:



DATA EXPLORATION

```
In [1]: %matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as m
import seaborn as sns
```

```
In [2]: loan=pd.read_csv('CleanedFile.csv',low_memory=False,encoding='ISO-8859-1',skipinitialspace=True)
loanfeature_df=loan
```

```
In [3]: reject=pd.read_csv('RejectLoan.csv',low_memory=False,encoding='ISO-8859-1',skipinitialspace=True)
rejectData_df=reject
```

```
In [4]: loan.head()
```

Out[4]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	sub_grade	emp_length	home_ownership
0	32000	32000	32000	36	13.99	1093.53	C	C3	10	MORTGAGE
1	35000	35000	35000	36	23.99	1372.97	E	E2	10	RENT
2	24000	24000	24000	60	13.99	558.32	C	C3	10	RENT
3	21000	21000	21000	36	13.49	712.54	C	C2	10	MORTGAGE
4	11000	11000	11000	36	9.49	352.32	B	B2	10	RENT

5 rows × 108 columns

```
In [5]: reject.head()
```

Out[5]:

	Amount Requested	Application Date	Loan Title	Risk_Score	Debt-To-Income Ratio	State	Employment Length	Policy Code
0	30000	2015-01-01	debt_consolidation	681	35.65	CA	1	0
1	5000	2015-01-01	debt_consolidation	648	10.62	CA	1	0
2	10000	2015-01-01	Debt consolidation	721	10.02	TX	7	0
3	10000	2015-01-01	major_purchase	659	19.05	AZ	1	0
4	5000	2015-01-01	debt_consolidation	501	10.73	IN	1	0

```
In [6]: summ_df = pd.DataFrame()
grouped = loan.groupby('issue_year')
summ_df = summ_df.append(grouped.aggregate(np.mean))
summ_df['loancount']=loan['loan_amnt'].groupby(loan['issue_year']).count()
summ_df['year']=summ_df.index
summ_df
```

Out[6]:

	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	emp_length	annual_inc
issue_year								
2007	8254.519071	7946.185738	1325.519071	36.000000	11.825108	264.842090	3.109453	64743.955224
2008	8825.428333	8347.273297	3179.903051	36.000000	12.061964	278.829465	4.326786	65241.740493
2009	9833.033516	9811.541375	8668.199773	36.000000	12.437247	329.842043	4.238023	69192.281954
2010	10528.240408	10078.262343	9817.588418	42.472362	11.985268	303.322235	4.884342	69512.984765
2011	12047.503568	11848.609640	11694.905161	44.419502	12.223365	338.436839	5.209060	69456.000506
2012	13461.709015	13452.932055	13429.753893	40.450840	13.637920	421.866695	5.638353	69720.227481
2013	14707.413733	14707.371267	14699.791847	42.122569	14.532012	452.393052	6.011334	73228.081364
2014	14870.156793	14870.156793	14865.334169	43.441427	13.770047	442.468144	5.877995	74854.143212
2015	15240.285862	15240.285862	15234.156411	43.860763	12.600242	441.857881	5.766651	76965.604424
2016	14734.039046	14733.974591	14727.754074	42.127636	13.043690	444.116471	5.739682	79498.270180

10 rows × 92 columns

```
In [7]: from __future__ import unicode_literals
import math

def plot_time_trends_1():
    year=loan['issue_year'].drop_duplicates()
    fico=loan['last_meanfico']
    dti=summ_df['dti']

    plt.figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='b')

    ax1=plt.subplot(211)
    plt.plot(['07','08','09','10','11','12','13','14','15','16'],dti,'r--')
    plt.xlabel('YEAR in 2000s')
    plt.ylabel('Avg. Debt to Income %')
    plt.legend(['DTI'])
    plt.grid(True)

    plt.title('Debt to Income Time Series')

    ax2=plt.subplot(212)
    bins=[650,660,670,680,690,700,710,720,730,740,750,760,770,780,790,800]
    plt.hist(fico, bins=bins, histtype='bar', rwidth=0.8 ,color='m')
    plt.xlabel('FICO')
    plt.ylabel('Counts')
    plt.title('FICO Distribution')

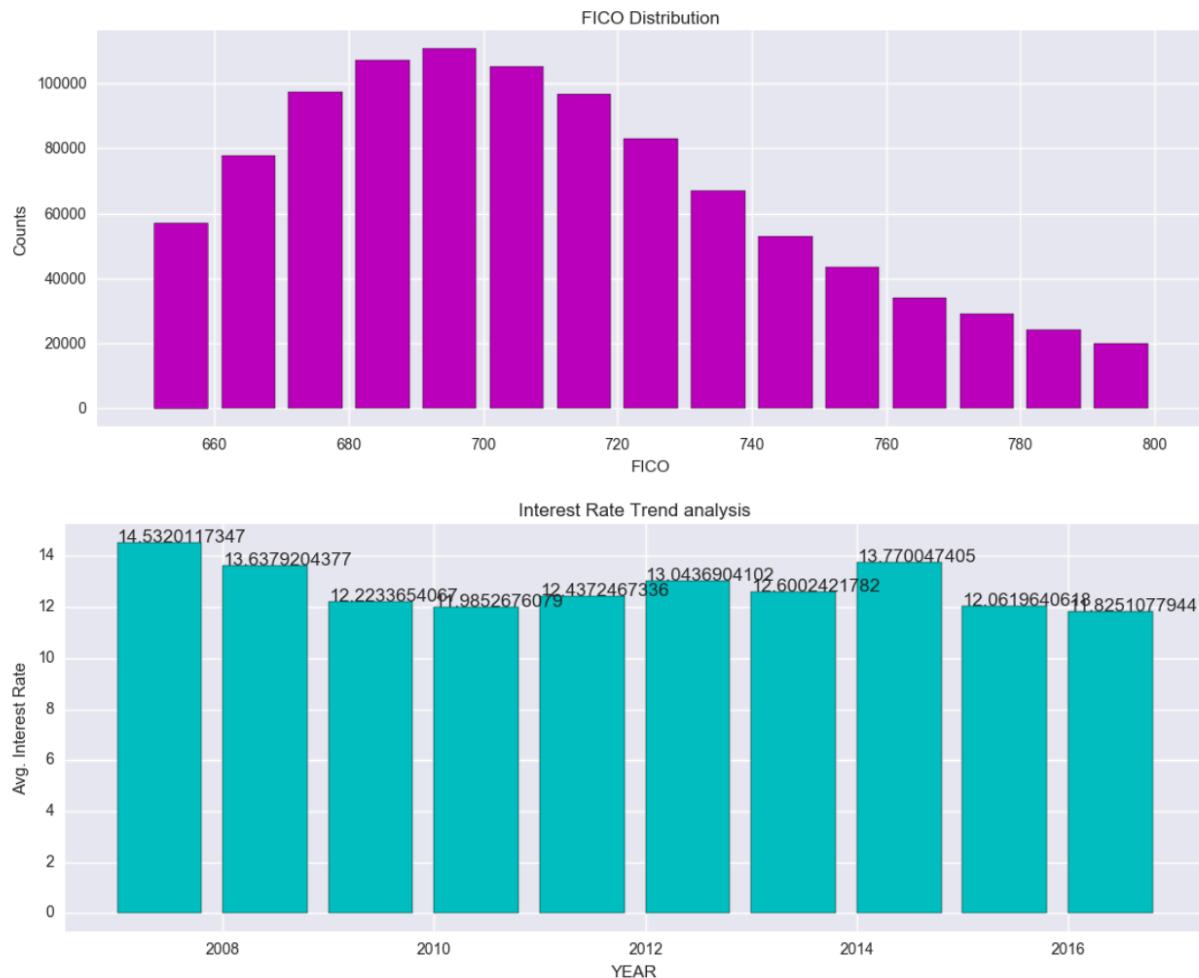
    ax2.margins(0.05)
    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                       wspace=0.35)
    plt.show()

def plot_time_trends_2():
    int_rt=summ_df['int_rate']
    year=loan['issue_year'].drop_duplicates()
    dti=summ_df['dti']

    plt.figure(num=None, figsize=(12, 10), dpi=80, facecolor='w', edgecolor='b')

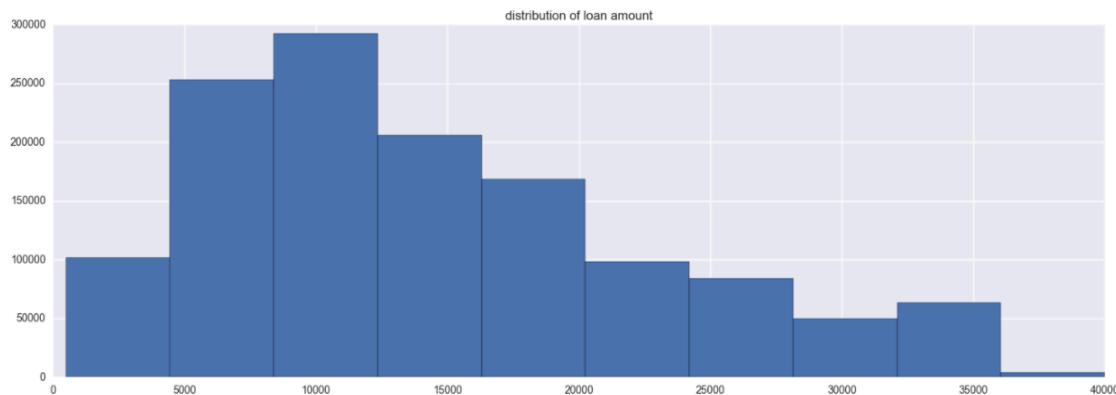
    ax4=plt.subplot(212)
    plt.bar(year,int_rt,color='c')
    plt.xlabel('YEAR')
    plt.ylabel('Avg. Interest Rate')
    plt.title('Interest Rate Trend analysis')
    for a,b in zip(year, int_rt):
        plt.text(a, b, str(b))

    ax4.margins(0.05)
    plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                       wspace=0.35)
    plt.show()
```



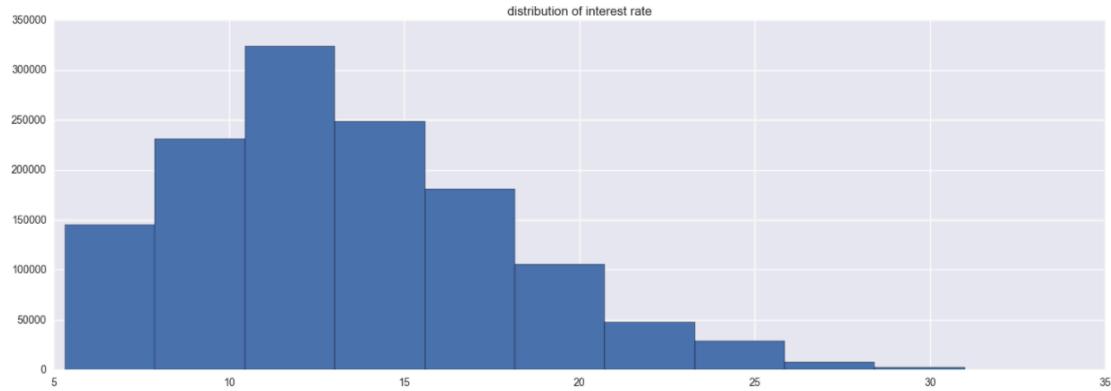
```
In [8]: plt.rc("figure", figsize=(18, 6))
loan["loan_amnt"].hist()
plt.title("distribution of loan amount")
```

```
Out[8]: <matplotlib.text.Text at 0x2018760ff60>
```



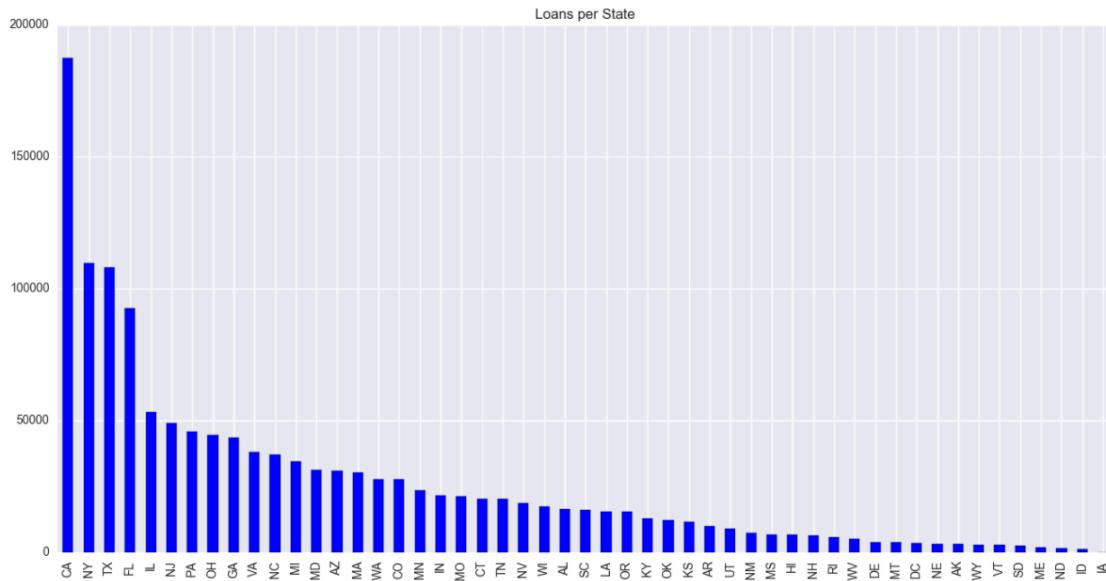
```
In [9]: plt.rc("figure", figsize=(18, 6))
loan["int_rate"].hist()
plt.title("distribution of interest rate")
```

```
Out[9]: <matplotlib.text.Text at 0x201878a0710>
```



```
In [10]: state_count = loan.addr_state.value_counts()
state_count.plot(kind = 'bar', figsize=(16,8), color='blue', title = 'Loans per State')
```

```
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x20187885978>
```



```
In [11]: loanpurp = pd.DataFrame()
loanpurp = loan.groupby(['term', 'purpose']).size().sort_values()
loa np = loanpurp.unstack()
loa np
```

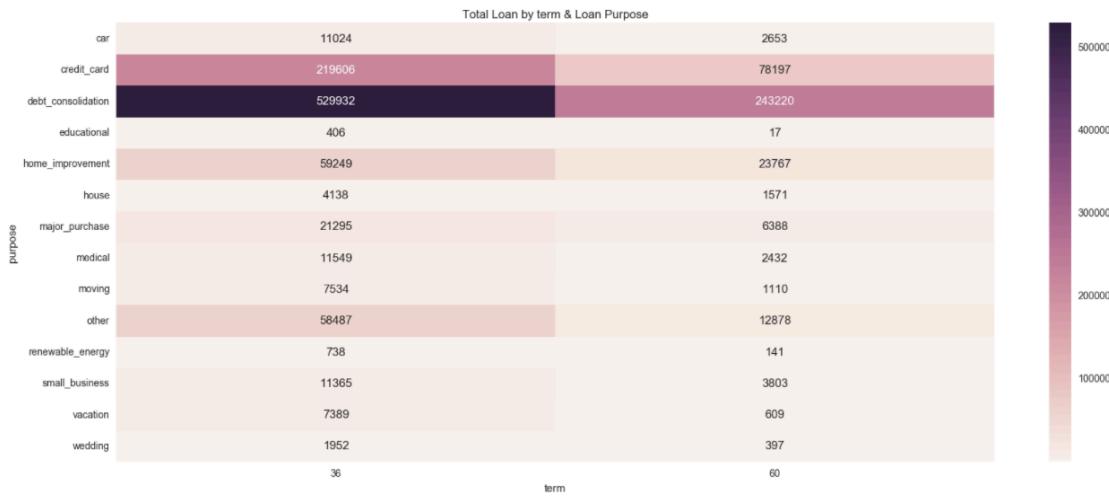
Out[11]:

purpose	car	credit_card	debt_consolidation	educational	home_improvement	house	major_purchase	medical	movi
term									
36	11024	219606	529932	406	59249	4138	21295	11549	7534
60	2653	78197	243220	17	23767	1571	6388	2432	1110

```
In [12]: import seaborn as sns
sns.set(style='white')
```

```
plt.figure(figsize=(20, 8))
plt.title('Total Loan by term & Loan Purpose')
ax = sns.heatmap(loa np.T, mask=loa np.T.isnull(), annot=True, fmt='g');
ax
```

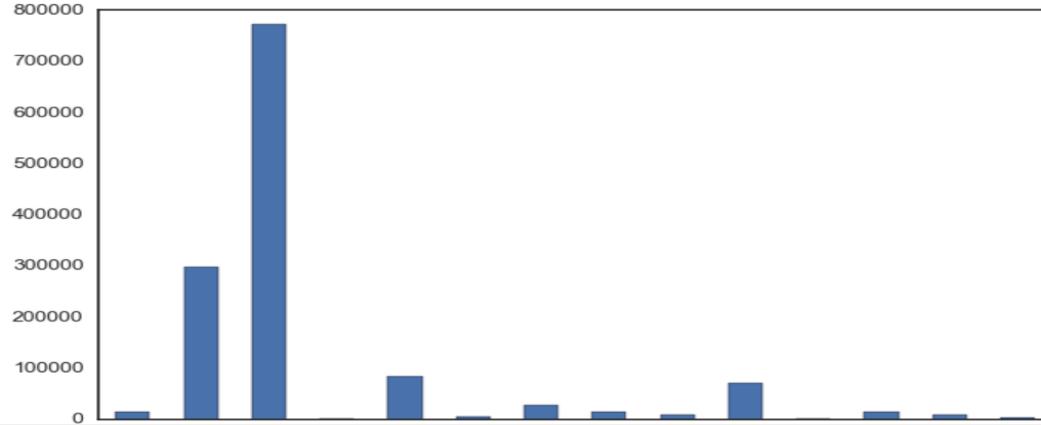
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x20187898f28>



```
In [13]: # What is the distribution of loans by purpose?
loans_by_purpose = loan.groupby('purpose')
print(loans_by_purpose['purpose'].count())
loans_by_purpose['purpose'].count().plot(kind='bar')
```

```
In [13]: # What is the distribution of loans by purpose?
loans_by_purpose = loan.groupby('purpose')
print(loans_by_purpose['purpose'].count())
loans_by_purpose['purpose'].count().plot(kind='bar')

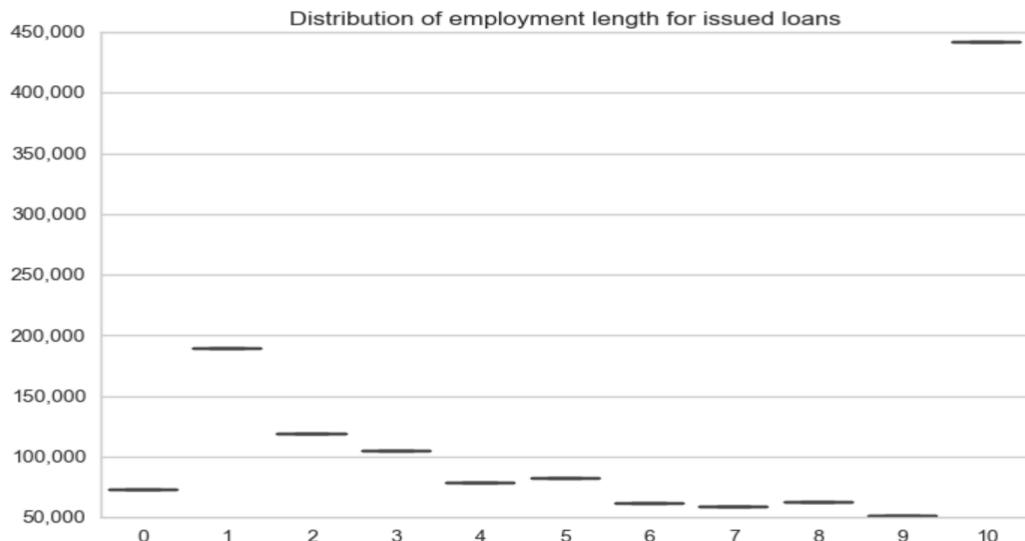
purpose
car                    13677
credit_card            297803
debt_consolidation     773152
educational             423
home_improvement        83016
house                   5709
major_purchase          27683
medical                  13981
moving                   8644
other                     71365
renewable_energy          879
small_business           15168
vacation                  7998
wedding                   2349
Name: purpose, dtype: int64
Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x201875d5240>
```



```
In [14]: s = pd.value_counts(loan['emp_length']).to_frame().reset_index()
s.columns = ['type', 'count']
def emp_dur_graph(graph_title):

    sns.set(style="whitegrid", color_codes=True)
    ax = sns.boxplot(y = "count", x = 'type', data=s)
    ax.set(xlabel = '', ylabel = '', title = graph_title)
    ax.get_yaxis().set_major_formatter(
        m.ticker.FuncFormatter(lambda x, p: format(int(x), ',')))
    _ = ax.set_xticklabels(ax.get_xticklabels(), rotation=0)

emp_dur_graph('Distribution of employment length for issued loans')
```



```
In [ ]: #Total interest rate ByYear
import pylab
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

pylab.rcParams['figure.figsize'] = (30.0, 10.0)

sns.set_style("darkgrid")

year=np.sort(loan["issue_year"].unique().tolist())

with plt.style.context('fivethirtyeight'):
    ax1 = plt.subplot2grid((4,2), (0,0), colspan=3)
    order=loan[ "purpose"].drop_duplicates().tolist()
    ax1 = sns.barplot(x='purpose',y="int_rate",data = loan, order = order)
    ax1 = plt.xticks(size = 10,rotation = 80)
    ax1 = plt.title("'purpose' v/s 'interest rate'")

    ax2 = plt.subplot2grid((4,2), (1, 0))
    ax2 = plt.rc("figure", figsize=(10, 8))
    ax2 = sns.barplot(x='term',y="int_rate",data = loan)
    ax2 = plt.title("'term' v/s 'interest rate'")

    ax3 = plt.subplot2grid((4,2), (1, 1))
    ax3 = plt.rc("figure", figsize=(6, 4))
    ax3 = sns.barplot(x='home_ownership',y="int_rate",data = loan)
    ax3 = plt.title("'home_ownership' v/s 'int_rate'")

    ax4 = plt.subplot2grid((4,2), (2, 0))
    ax4 = plt.rc("figure", figsize=(10, 8))
    ax4 = sns.barplot(x='emp_length',y="int_rate",data = loan)
    ax4 = plt.title("'employee length' v/s 'interest rate'")
    ax4 = plt.xticks(size = 10,rotation = 80)

    ax5 = plt.subplot2grid((4,2), (2, 1))
    ax5 = plt.rc("figure", figsize=(10, 8))
    order = np.sort(loan[ "issue_year"].unique().tolist())
    ax5 = sns.barplot(x='issue_year',y="int_rate",data = loan, order = order)
    ax5 = plt.title("'issue_year' v/s 'interest rate'")
    ax5 = plt.xticks(size = 10,rotation = 80)

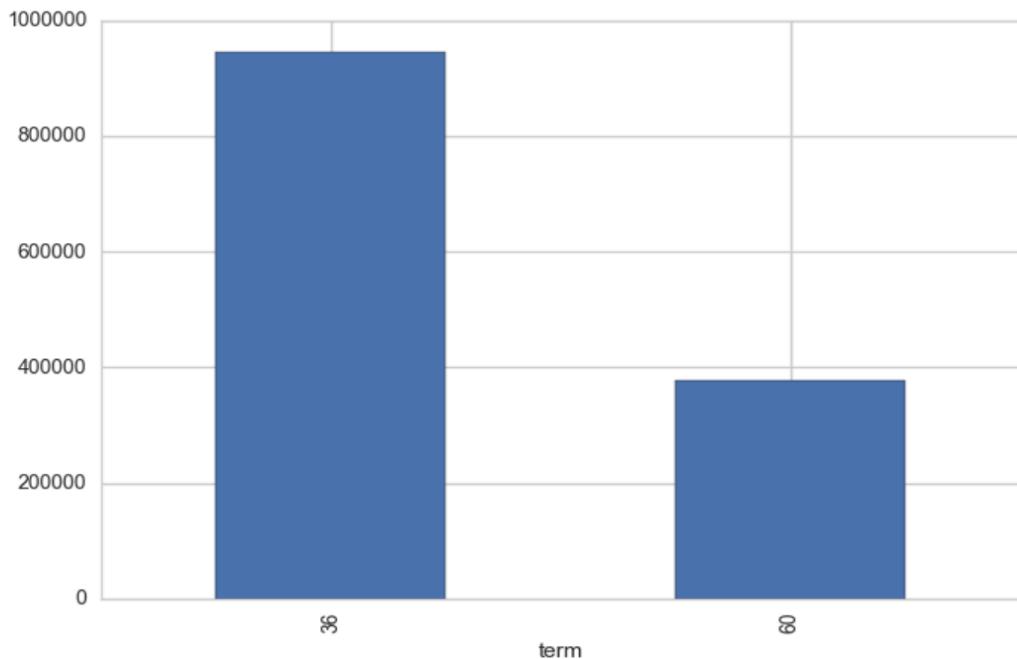
    ax6 = plt.subplot2grid((4,2), (3, 0))
    ax6 = plt.rc("figure", figsize=(10, 8))
    ax6 = sns.barplot(x='application_type',y="int_rate",data = loan)
    ax6 = plt.title("'application type' v/s 'interest rate'")

plt.tight_layout(2)
```

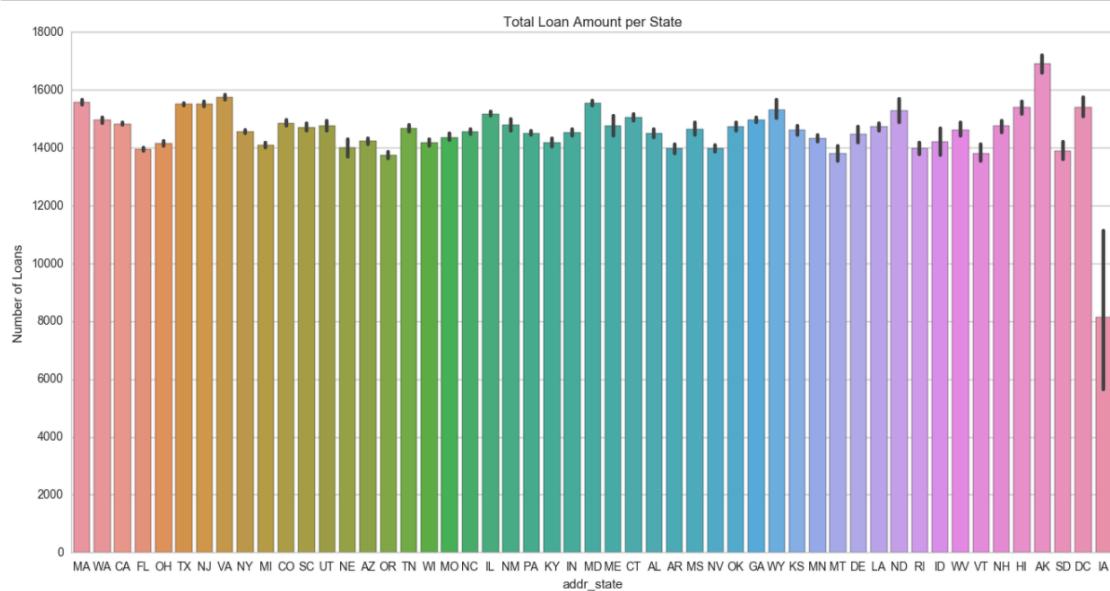
```
In [15]: # What is the distribution of loans by term?
loans_by_term = loan.groupby('term')
print(loans_by_term['term'].count())
loans_by_term['term'].count().plot(kind='bar')
```

```
term
36      944664
60      377183
Name: term, dtype: int64
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x201875f5f60>
```

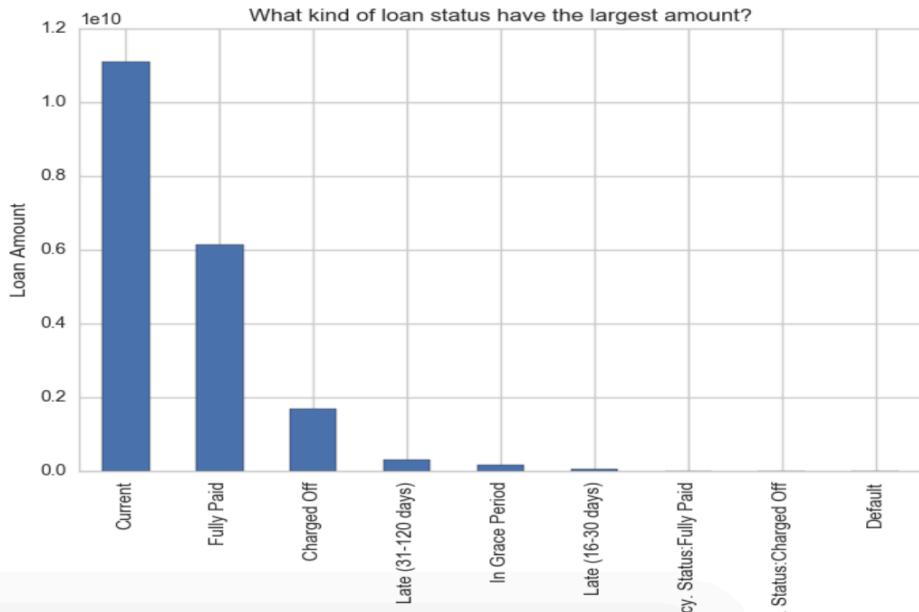


```
In [16]: fig, ax = plt.subplots(figsize = (16,8))
ax = sns.barplot(x='addr_state', y='loan_amnt', data=loan)
ax.set(ylabel = 'Number of Loans', title = 'Total Loan Amount per State')
plt.show()
```



```
In [17]: loan.groupby('loan_status')['loan_amnt'].sum().sort_values(ascending=0).plot(kind='bar')
plt.xlabel('Loan Status')
plt.ylabel('Loan Amount')
plt.title('What kind of loan status have the largest amount?')
```

Out[17]: <matplotlib.text.Text at 0x20188c772b0>



```
In [18]: rejectData_df['issue_d'] = rejectData_df['Application Date'].str.split("-")
rejectData_df['Issue_Year']=rejectData_df['issue_d'].str[0]
reject_cols_to_keep=['Amount Requested', 'Employment Length', 'Risk_Score', 'Debt-To-Income Ratio', 'Issue_Year']
rejectData_df=rejectData_df[reject_cols_to_keep]
rejectData_df['Approval_Status']=0
rejectData_df.head()
```

C:\Users\vishakha\Anaconda3\lib\site-packages\ipykernel_main_.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>

Out[18]:

	Amount Requested	Employment Length	Risk_Score	Debt-To-Income Ratio	Issue_Year	Approval_Status
0	30000	1	681	35.65	2015	0
1	5000	1	648	10.62	2015	0
2	10000	7	721	10.02	2015	0
3	10000	1	659	19.05	2015	0
4	5000	1	501	10.73	2015	0

```
In [19]: cols_to_keep=['loan_amnt', 'emp_length', 'last_meanfico', 'dti','issue_year']
loanfeature_df=loanfeature_df[cols_to_keep]
loanfeature_df['Approval_Status']=1
loanfeature_df.columns=['Amount Requested', 'Employment Length', 'Risk_Score', 'Debt-To-Income Ratio', 'Issue_Year', 'Approval_Status']
loanfeature_df.head()
```

C:\Users\vishakha\Anaconda3\lib\site-packages\ipykernel_main_.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>
app.launch_new_instance()

Out[19]:

	Amount Requested	Employment Length	Risk_Score	Debt-To-Income Ratio	Issue_year	Approval_status
0	32000	10	712.0	17.30	2016	1
1	35000	10	672.0	15.61	2016	1
2	24000	10	697.0	19.30	2016	1
3	21000	10	697.0	25.00	2016	1
4	11000	10	682.0	11.90	2016	1

```
In [20]: x_train1=loanfeature_df
x_test1=rejectData_df
frames=[x_train1,x_test1]
result = pd.concat(frames)
result.head()
```

Out[20]:

	Amount Requested	Employment Length	Risk_Score	Debt-To-Income Ratio	Issue_year	Approval_status
0	32000	10	712.0	17.30	2016	1
1	35000	10	672.0	15.61	2016	1
2	24000	10	697.0	19.30	2016	1
3	21000	10	697.0	25.00	2016	1
4	11000	10	682.0	11.90	2016	1

```
In [ ]: # Analysis of Yearly approved and rejected loans
y=result
y=pd.DataFrame({'count' : y.groupby( ['Issue_year','Approval_status'] ).size()}).reset_index()
y
```

```
In [ ]: # Barplot to show yearwise approval status
g = sns.factorplot(x="Issue_year", y="count", hue="Approval_status", data=y,
                    size=10, kind="bar", palette="muted")
g.despine(left=True)
g.set_ylabels("Count")
```

```
In [ ]: # Barplot to show yearwise approval status
g = sns.factorplot(x="Issue_year", y="count", hue="Approval_status", data=y,
                    size=10, kind="bar", palette="muted")
g.despine(left=True)
g.set_ylabels("Count")
```

```
In [ ]: # Analysis of approval status
X=result.groupby(['Approval_status']).count()

X['status']=X.index
# print(X)

# Barplot to show categorization of approval status
g=sns.factorplot(x="status",y='Issue_year',data=X,size=6, kind="bar", palette="muted")
g.despine(left=True)
g.set_ylabels("Count")
```

```
In [ ]: # Plot to show Risk_Score analysis as per the approval status
z=result
g = sns.FacetGrid(z,col="Approval_status",col_order=[0,1])
g = g.map(plt.hist, "Risk_Score",color="m")
```

Part2:

CLASSIFICATION

```
In [1]: # Package imports
%matplotlib inline
from IPython.display import Image
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn
import warnings
from sklearn.metrics import roc_curve
from sklearn.utils import ConvergenceWarning
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn import cross_validation
from sklearn import tree
from sklearn import svm
from sklearn import ensemble
from sklearn import neighbors
from sklearn import linear_model
from sklearn import metrics
from sklearn import preprocessing
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
import pandas as pd
import statsmodels.formula.api as smf
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split, KFold, StratifiedShuffleSplit, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.feature_selection import RFE
import warnings
warnings.filterwarnings('ignore')
```

```
In [2]: from io import StringIO
import requests
import json
import pandas as pd

# @hidden_cell
# This function accesses a file in your Object Storage. The definition contains your credentials.
# You might want to remove those credentials before you share your notebook.
def get_object_storage_file_with_credentials_14d326a3558e4fc0bfeb2361ed8fd35c(container, filename):
    """This functions returns a StringIO object containing
    the file content from Bluemix Object Storage."""
    url1 = ''.join(['https://identity.open.softlayer.com', '/v3/auth/tokens'])
    data = {'auth': {'identity': {'methods': ['password'],
                                  'password': {'user': {'name': 'member_3c25b61fef62727472d70bdd15a6ecc3b9af4f3b', 'domain': {'id': 'fcf7d2f57ba04101a6ee7e43c6b09dc5'},
                                  'password': 'VW28-eGXM{X-!mji'}}}}}
    headers1 = {'Content-Type': 'application/json'}
    resp1 = requests.post(url=url1, data=json.dumps(data), headers=headers1)
    resp1_body = resp1.json()
    for e1 in resp1_body['token']['catalog']:
        if(e1['type']=='object-store'):
            for e2 in e1['endpoints']:
                if(e2['interface']=='public' and e2['region']=='dallas'):
                    url2 = ''.join([e2['url'],'/', container, '/', filename])
                    s_subject_token = resp1.headers['x-subject-token']
                    headers2 = {'X-Auth-Token': s_subject_token, 'accept': 'application/json'}
                    resp2 = requests.get(url=url2, headers=headers2)
    return StringIO(resp2.text)

df_data_1 = pd.read_csv(get_object_storage_file_with_credentials_14d326a3558e4fc0bfeb2361ed8fd35c(
    'ADSAssignment2', 'Final_loan.csv'))
df_data_1.head()
```

Out[2]:

	loan_amnt	last_meanfico	dti	addr_state	emp_length	policy_code	purpose	issue_month	issue_year	status
0	5000	737	27.65	AZ	10	1	credit_card	12	2011	1
1	2500	742	1.00	GA	1	1	car	12	2011	1
2	2400	737	8.72	IL	10	1	small_business	12	2011	1
3	10000	692	20.00	CA	10	1	other	12	2011	1
4	3000	697	17.94	OR	1	1	other	12	2011	1

In [3]: #Keep the following 6 features (variables) which are important
 cols_to_keep = ['loan_amnt', 'last_meanfico', 'dti', 'addr_state', 'emp_length', 'purpose']
 train, test = sklearn.cross_validation.train_test_split(df_data_1, train_size = 0.7)
 train_y = train['status']
 test_y = test['status']
 train_x = train[cols_to_keep]
 test_x = test[cols_to_keep]

In [4]: from sklearn import preprocessing
 # Discreet value integer encoder
 label_encoder = preprocessing.LabelEncoder()
 # State is string and we want discrete integer values
 train_x['state'] = label_encoder.fit_transform(train_x['addr_state'])
 test_x['state'] = label_encoder.fit_transform(test_x['addr_state'])
 train_x['purp'] = label_encoder.fit_transform(train_x['purpose'])
 test_x['purp'] = label_encoder.fit_transform(test_x['purpose'])

In [5]: train_x = train_x._get_numeric_data()
 test_x = test_x._get_numeric_data()

In [6]: train_x.head()

Out[6]:

	loan_amnt	last_meanfico	dti	emp_length	state	purp
6666991	5000	501	13.97	1	37	55
460892	10000	697	3.72	10	5	49
6289137	7000	304	10.60	1	20	45
12054109	25000	304	69.76	1	2	45
11516824	1500	304	319.24	1	36	52

In [8]: def calculate_roc_curve(Y_test, y_pred, pos_label):
 fpr, tpr, _ = roc_curve(Y_test, y_pred)
 #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()

 def calculate_confusion_matrix(y_true, y_pred):
 cm=confusion_matrix(y_true, y_pred)
 print(cm)

In [9]: #logistic regression
 log_reg = LogisticRegression()
 log_reg.fit(train_x, train_y)
 print ("Intercept is ",log_reg.intercept_)
 print("Coefficient is ",log_reg.coef_)
 y_pred=log_reg.predict(test_x)

 #calculate ROC curve
 preds = log_reg.predict_proba(test_x)[:,1]
 calculate_roc_curve(test_y, preds,2)

 #calculate Confusion Matrix
 calculate_confusion_matrix(test_y, y_pred)

 print(accuracy_score(test_y, y_pred))

```
('Intercept is ', array([-4.35235659]))
('Coefficient is ', array([[ -2.07088278e-05,   6.27878832e-03,  -2.32898171e-02,
   3.20780048e-01,  -1.67046429e-02,  -2.21743219e-02]]))

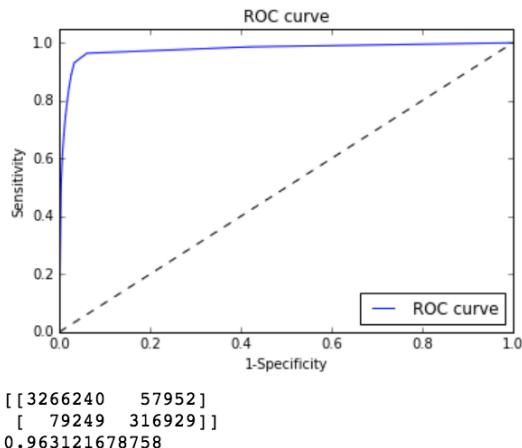
ROC curve
Sensitivity
1-Specificity
[ 0.0  0.2  0.4  0.6  0.8  1.0]
[ 0.0  0.2  0.4  0.6  0.8  1.0]

[[3256338  67854]
 [ 213655  182523]]
0.924333063647
```

```
In [10]: #Random Forest
rf = RandomForestClassifier(n_estimators=2)
rf.fit(train_x, train_y)
preds = rf.predict_proba(test_x)[:,1]
y_pred=rf.predict(test_x)

#calculate ROC curve
calculate_roc_curve(test_y, y_pred,2)
#calculate Confusion Matrix

calculate_confusion_matrix(test_y, y_pred)
print(accuracy_score(test_y, y_pred))
```



```
In [ ]: print("Starting Support Vector Machine")
clf = SVC()
clf.fit(train_x, train_y)
y_pred=clf.predict(test_x)

#calculate ROC curve
preds = clf.predict_proba(test_x)[:,1]
calculate_roc_curve(test_y, preds,2)
#calculate Confusion Matrix

calculate_confusion_matrix(Y_test, y_pred)
print(accuracy_score(Y_test, y_pred))
```

```
In [12]: # RFE Code to extract features
model = LogisticRegression()
rfe = RFE(model, 5)
fit = rfe.fit(train_x, train_y)
print("Num Features:", fit.n_features_)
print("Selected Features" , fit.support_)
print("Feature Ranking: ", fit.ranking_)

('Num Features:', 5)
('Selected Features', array([False, True, True, True, True], dtype=bool))
('Feature Ranking: ', array([2, 1, 1, 1, 1]))
```

```
In [13]: #stats model for feature extraction
import statsmodels.api as sm
from statsmodels.formula.api import logit, probit, poisson, ols
logit = sm.Logit(train_y, train_x)
affair_mod = logit.fit()
print(affair_mod.summary())
```

Optimization terminated successfully.
 Current function value: 0.266703
 Iterations 13

Logit Regression Results						
Dep. Variable:	status	No. Observations:	8680863			
Model:	Logit	Df Residuals:	8680857			
Method:	MLE	Df Model:	5			
Date:	Fri, 14 Apr 2017	Pseudo R-squ.:	0.2142			
Time:	02:12:54	Log-Likelihood:	-2.3152e+06			
converged:	True	LL-Null:	-2.9464e+06			
		LLR p-value:	0.000			
	coef	std err	z	P> z	[95.0% Conf. Int.]	
loan_amnt	-4.056e-05	1.29e-07	-315.473	0.000	-4.08e-05	-4.03e-05
last_meanfico	0.0032	6.23e-06	516.059	0.000	0.003	0.003
dti	-0.0336	9.24e-05	-363.547	0.000	-0.034	-0.033
emp_length	0.3261	0.000	840.105	0.000	0.325	0.327
state	-0.0316	8.06e-05	-392.551	0.000	-0.032	-0.031
purp	-0.0644	6.83e-05	-943.143	0.000	-0.065	-0.064

```
In [ ]: # build a neural network

#Calculating rows and columns for input dfs
trn_rows,trn_cols=train_x.shape
tst_rows,tst_cols=test_x.shape

# build train dataset
train_data = ClassificationDataSet(trn_cols, 1 , nb_classes=2)
for k in range(len(train_x)):
    train_data.addSample(train_x.iloc[k],train_y.iloc[k])

# build test dataset
test_data = ClassificationDataSet(tst_cols, 1 , nb_classes=2)
for k in range(len(test_x)):
    test_data.addSample(test_x.iloc[k],test_y.iloc[k])

print("Train Dataset input length: {}".format(len(train_data['input'])))
print("Train Dataset output length: {}".format(len(train_data['target'])))
print("Train Dataset input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))
))

print("Train Data length: {}".format(len(train_data)))
print("Test Data length: {}".format(len(test_data)))

# encode with one output neuron per class
train_data._convertToOneOfMany()
test_data._convertToOneOfMany()

print("Train Data input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))
print("Test Data input|output dimensions are {}|{}".format(test_data.indim, test_data.outdim))

# build network (INPUT=10,HIDDEN=5,CLASSES=2,outclass=SoftmaxLayer)
print("Building Neural network with 5 hidden layer")
network = buildNetwork(train_data.indim,5,train_data.outdim,outclass=SoftmaxLayer)

# train network
print("Training the network, it may take a while...")
trainer = BackpropTrainer(network,dataset=train_data,momentum=0.1,verbose=True,weightdecay=0.01)
trainer.trainOnDataset(train_data, 1) #training model on One epoch

print("Total epochs: {}".format(trainer.totalePOCHS))

# test network
print("Predicting the output array with the trained model")
output = network.activateOnDataset(test_data).argmax(axis=1)

#Neural network Percent error and accuracy
print("Percent error: {}".format(percentError(output, test_data['class'])))
print("Model Accuracy: {}".format(Validator.classificationPerformance(output, test_y)))

#Compute confusion metrics
calculate_confusion_matrix(test_y,output)
```

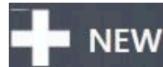
AZUAR ML

1. Importing Dataset

- Click 'Datasets' on the left menu of the studio.



- Now click on the '+' to import the dataset from the local storage.



- When NEW is click it will give following option:



- choose your dataset and follow the instructions to finish import.

2. Feature selection experiment

Before we proceed with our experiment, we have done feature selection in a separate experiment.

Since we have 10 different features which consist of 5 Main features and 3 derived feature, we apply the feature selection technique to come up with the best feature used to classify the applicant Loan request.



We used 2 different method to find the best feature for our model:

- Pearson Correlation
- Chi-Squared

emp_length	last_meanfico	purpose	dti	loan_amnt
5178164.458377	3979911.756731	1328207.627503	810644.177579	542349.419562

So, we shortlist the above features to use it in our clustering experiment.

3. Drag dataset to classification experiment

```
In [3]: #Keep the following 6 features (variables) which are important
cols_to_keep = ['loan_amnt', 'last_meanfico', 'dti','addr_state','emp_length','purpose']
train, test = sklearn.cross_validation.train_test_split(df_data_1, train_size = 0.7)
train_y = train['status']
test_y = test['status']
train_x = train[cols_to_keep]
test_x = test[cols_to_keep]

In [4]: from sklearn import preprocessing
# Discrete value integer encoder
label_encoder = preprocessing.LabelEncoder()
# State is string and we want discrete integer values
train_x['state'] = label_encoder.fit_transform(train_x['addr_state'])
test_x['state'] = label_encoder.fit_transform(test_x['addr_state'])
train_x['purp'] = label_encoder.fit_transform(train_x['purpose'])
test_x['purp'] = label_encoder.fit_transform(test_x['purpose'])

In [5]: train_x = train_x._get_numeric_data()
test_x = test_x._get_numeric_data()
```

Logistic Regression:

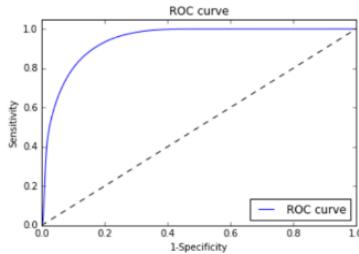
```
#Logistic regression
log_reg = LogisticRegression()
log_reg.fit(train_x, train_y)
print ("Intercept is ",log_reg.intercept_)
print("Coefficient is ",log_reg.coef_)
y_pred=log_reg.predict(test_x)

#calculate ROC curve
preds = log_reg.predict_proba(test_x)[:,1]
calculate_roc_curve(test_y, preds,2)

#calculate Confusion Matrix
calculate_confusion_matrix(test_y, y_pred)

print(accuracy_score(test_y, y_pred))
```

('Intercept is ', array([-4.35235659]))
('Coefficient is ', array([[-2.07088278e-05, 6.27878832e-03, -2.32898171e-02, 3.20780048e-01, -1.67046429e-02, -2.21743219e-02]]))



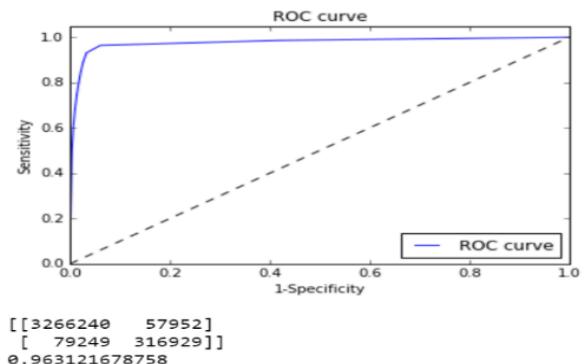
Sensitivity
1-Specificity

[[3256338 67854]
[213655 182523]]
0.924333063647

Random Dense Forest:

```
#Random Forest
rf = RandomForestClassifier(n_jobs=2)
rf.fit(train_x, train_y)
preds = rf.predict_proba(test_x)[:,1]
y_pred=rf.predict(test_x)
#calculate ROC curve
calculate_roc_curve(test_y, y_pred,2)
#calculate Confusion Matrix

calculate_confusion_matrix(test_y, y_pred)
print(accuracy_score(test_y, y_pred))
```



Neural Network models:

```
In [ ]: # build a neural network

#Calculating rows and columns for input dfs
trn_rows,trn_cols=train_x.shape
tst_rows,tst_cols=test_x.shape

# build train dataset
train_data = ClassificationDataSet(trn_cols, 1 , nb_classes=2)
for k in range(len(train_x)):
    train_data.addSample(train_x.iloc[k],train_y.iloc[k])

# build test dataset
test_data = ClassificationDataSet(tst_cols, 1 , nb_classes=2)
for k in range(len(test_x)):
    test_data.addSample(test_x.iloc[k],test_y.iloc[k])

print("Train Dataset input length: {}".format(len(train_data['input'])))
print("Train Dataset output length: {}".format(len(train_data['target'])))
print("Train Dataset input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))

print("Train Data length: {}".format(len(train_data)))
print("Test Data length: {}".format(len(test_data)))

# encode with one output neuron per class
train_data._convertToOneOfMany()
test_data._convertToOneOfMany()

print("Train Data input|output dimensions are {}|{}".format(train_data.indim, train_data.outdim))
print("Test Data input|output dimensions are {}|{}".format(test_data.indim, test_data.outdim))

# build network (INPUT=10,HIDDEN=5,CLASSES=2,outclass=SoftmaxLayer)
print("Building Neural network with 5 hidden layer")
network = buildNetwork(train_data.indim,5,train_data.outdim,outclass=SoftmaxLayer)

# train network
print("Training the network, it may take a while...")
trainer = BackpropTrainer(network,dataset=train_data,momentum=0.1,verbose=True,weightdecay=0.01)
trainer.trainOnDataset(train_data, 1) #training model on One epoch

print("Total epochs: {}".format(trainer.totalePOCHS))

# test network
print("Predicting the output array with the trained model")
output = network.activateOnDataset(test_data).argmax(axis=1)

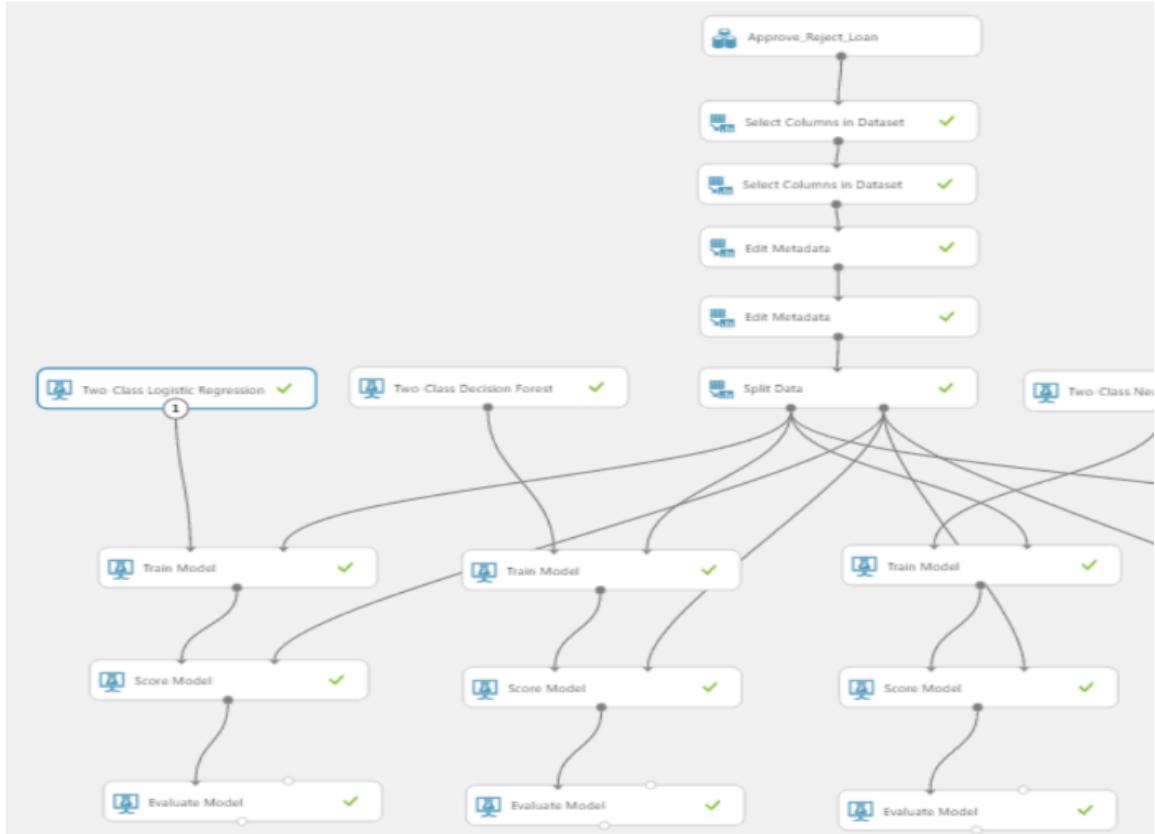
#Neural network Percent error and accuracy
print("Percent error: {}".format(percentError(output, test_data['class'])))
print("Model Accuracy: {}".format(Validator.classificationPerformance(output, test_y)))

#Compute confusion metrics
calculate_confusion_matrix(test_y,output)
```

After performing our analysis on Jupyter notebook, we move to Azure ML platform to run Neural network following Dense Random Forest and Logistic classification.

4. Select Columns in dataset

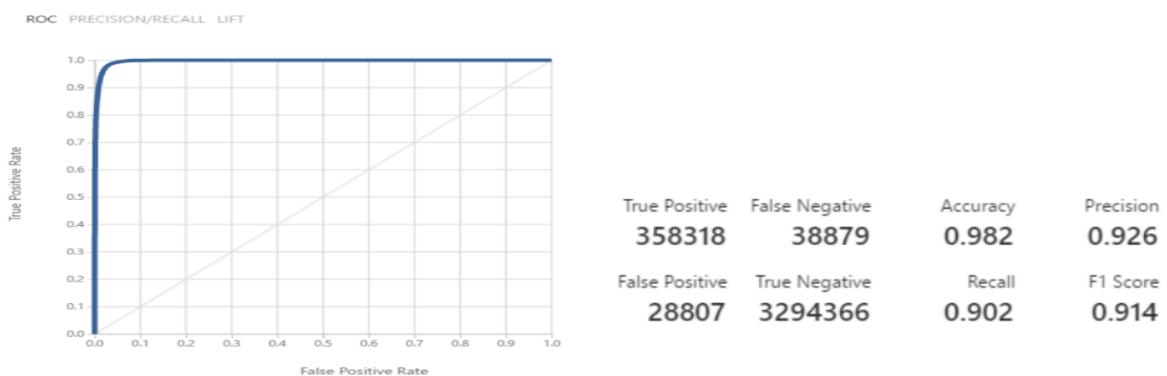
5. Split data: testing and validation



We implemented the following four classification algorithm for classifying user request:

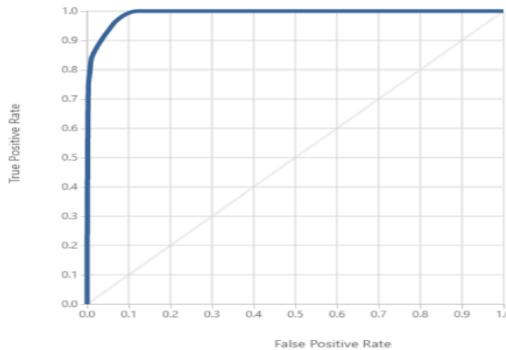
- Logistic Regression
- Random Dense Forest
- Neural Network

Two Class Decision Forest:



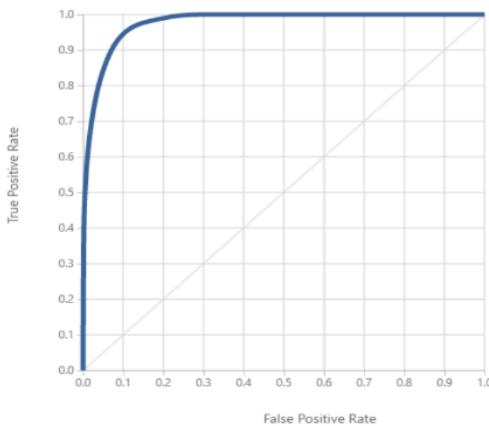
Neural Network:

The ROC curve is shown is visualized as below, having a good accuracy of 97%.



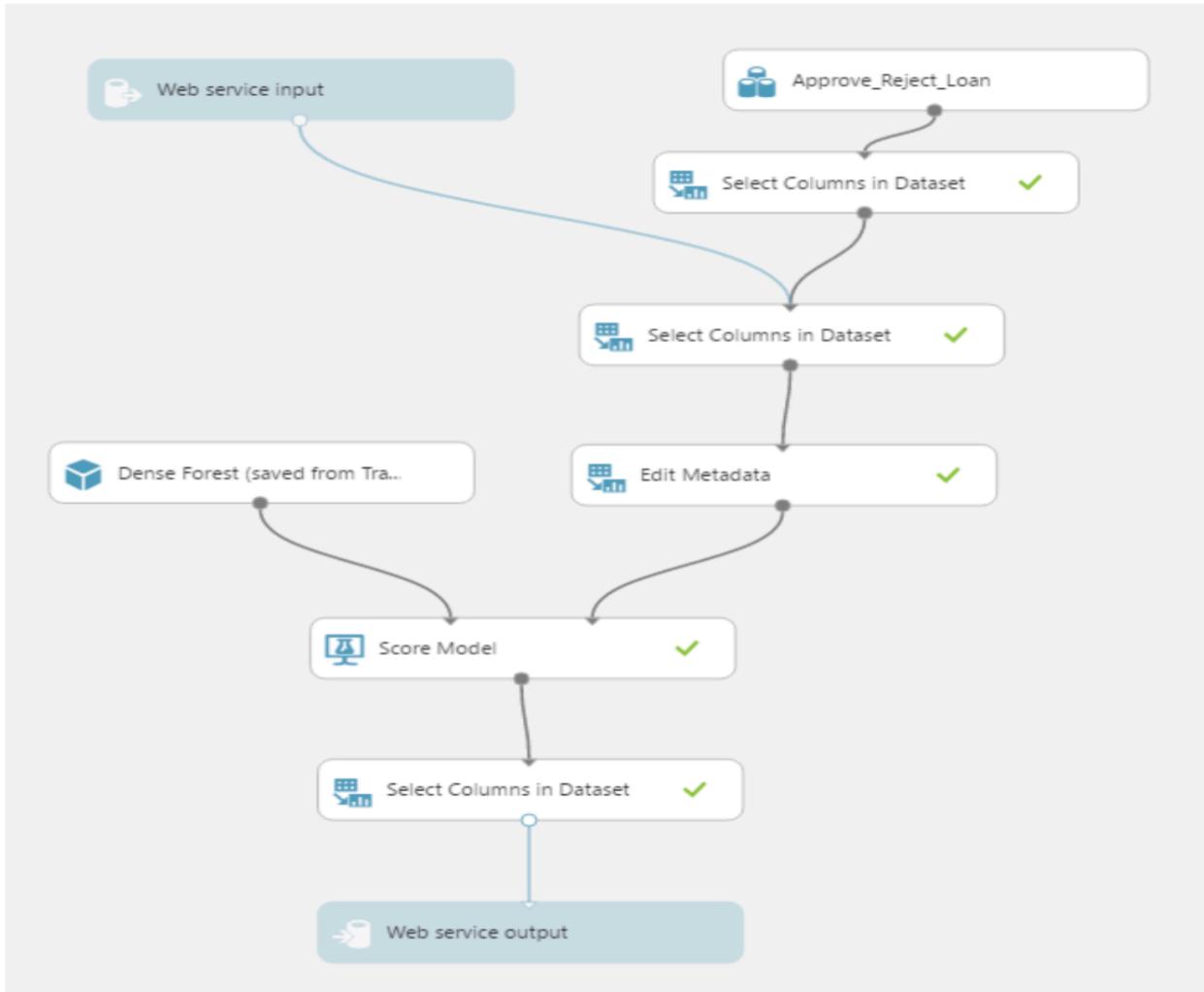
True Positive	False Negative	Accuracy	Precision
306973	90224	0.971	0.950
False Positive	True Negative	Recall	F1 Score
16295	3306867	0.773	0.852

Logistic Regression:



True Positive	False Negative	Accuracy	Precision
262769	134428	0.949	0.825
False Positive	True Negative	Recall	F1 Score
55759	3267403	0.662	0.734

6. Setup web service: classification



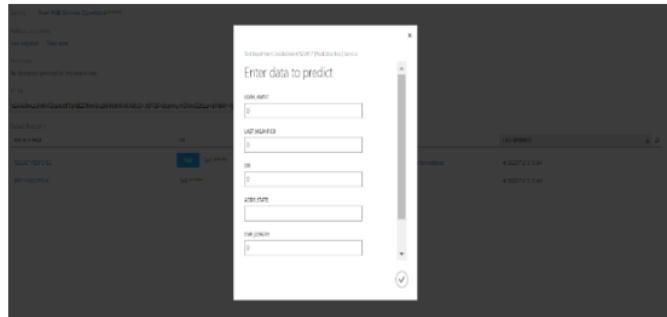
When we visualize the Score Model component we get scored labels and scored probabilities along with other features.

loan_amnt	last_meanfico	dti	addr_state	emp_length	purpose	Scored Labels	Scored Probabilities
5000	737	27.65	AZ	10	credit_card	1	0.994397
2500	742	1	GA	1	car	0	0.28303
2400	737	8.72	IL	10	small_business	1	0.924646
10000	692	20	CA	10	other	1	0.677337
3000	697	17.94	OR	1	other	0	0.339034
5000	732	11.2	AZ	3	wedding	1	0.835878

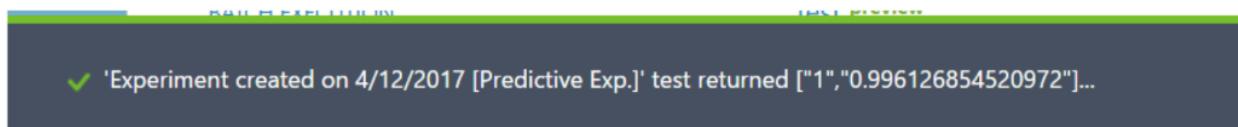
Scored Probabilities: We will consider taking this value as an output of web service to allow user to choose a model based on accuracy instead of using his own judgment.

We are using 'Select Column in Dataset' component to extract just these 2 columns.

Scored Labels	Scored Probabilities
1	0.994397
0	0.28303
1	0.924646
1	0.677337



The output of this experiment is JSON format :



7. CLUSTERING

K-MEANS Clustering:

▲ K-Means Clustering

Create trainer mode

Single Parameter ▾

Number of Centroids

5

Initialization

K-Means++ Fast ▾

Random number seed

Metric

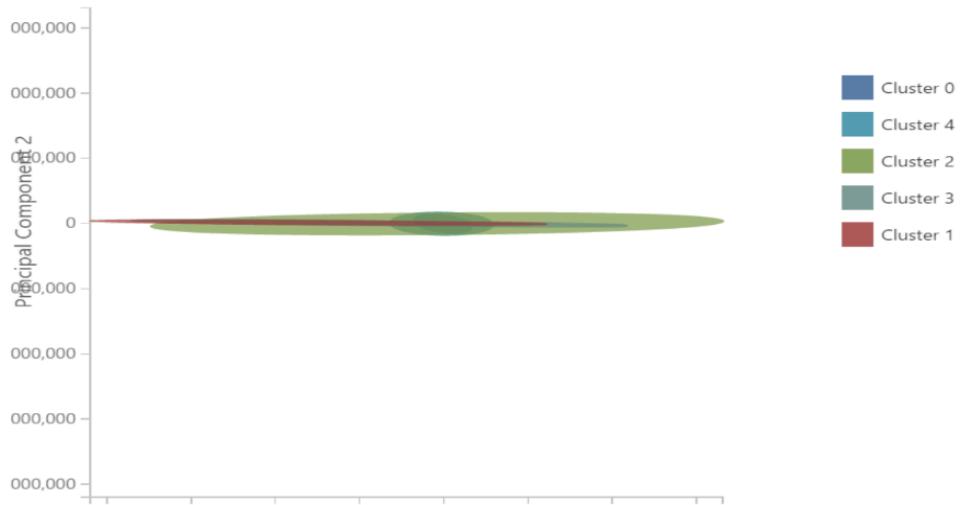
Euclidean ▾

We have selected columns as below for clustering

loan_amnt ✕	term ✕	emp_length ✕
home_ownership ✕	annual_inc ✕	
verification_status ✕	purpose ✕	addr_state ✕
dti ✕	delinq_2yrs ✕	fico_range_high ✕
inq_last_6mths ✕	mths_since_last_delinq ✕	
open_acc ✕	pub_rec ✕	revol_util ✕
application_type ✕	pub_rec_bankruptcies ✕	total_acc ✕
open_acc_6m ✕	mths_since_last_major_derog ✕	

Splitting the data follow what we did in previous experiments 70% 30% split for test and train.

Finally, after the training is done we have assigned below columns to assign to cluster.

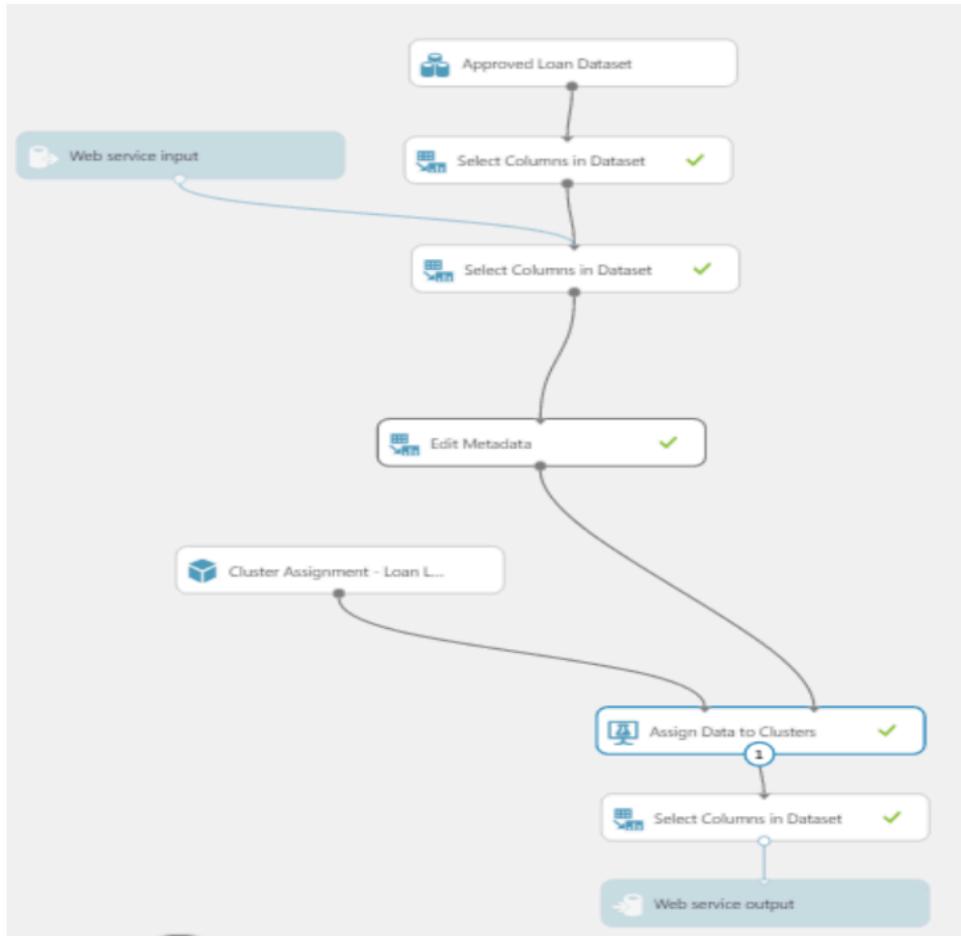


Manual Clustering:

For Manual clustering, we choose Term and Fico Score as our main factor to cluster the user and predict their interest rate.

We first divide the user based on the Fico score bins and then split them on the Term (36 or 60 months) and created 6 csv file to train our model and predict our interest rate.

8. Setup web service: clustering



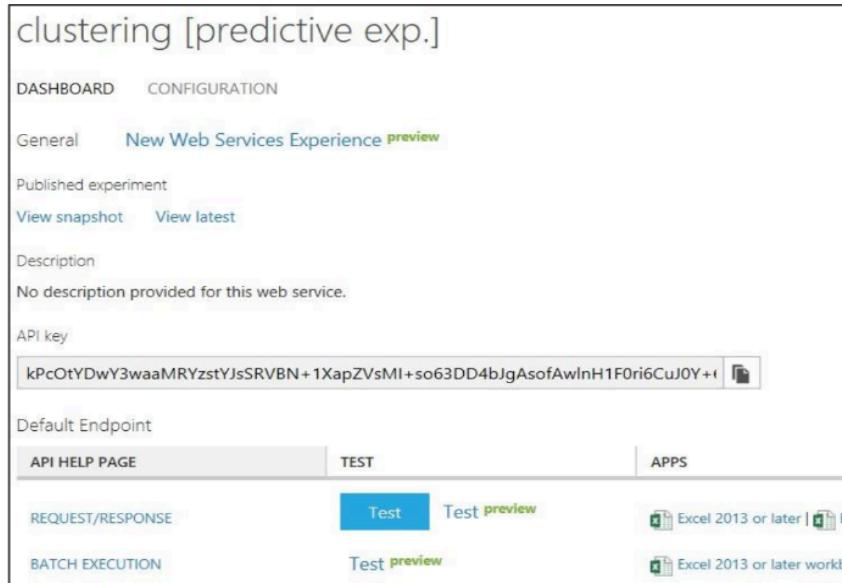
The Output is shown as:

rows	columns
1321847	1



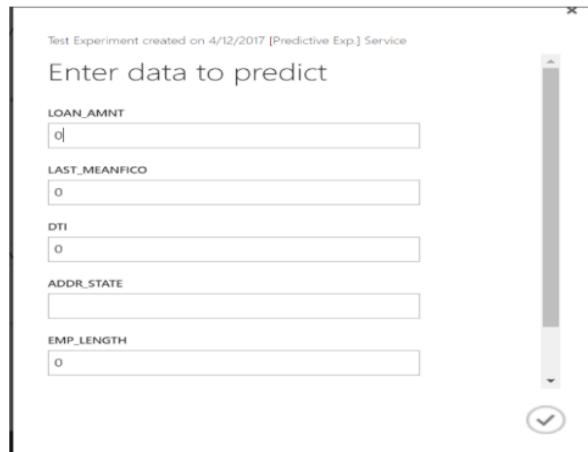
9. Deploy the web service

Like what we did for classification we can Test out web service.



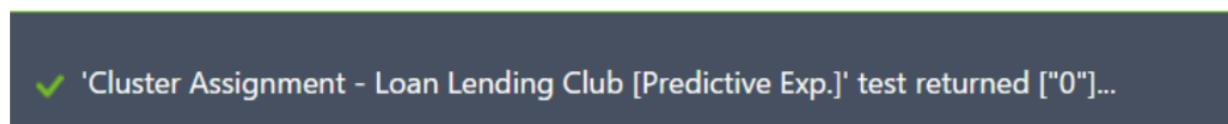
The screenshot shows the 'clustering [predictive exp.]' experiment configuration page. At the top, there are 'DASHBOARD' and 'CONFIGURATION' tabs. Below them, there are 'General' and 'New Web Services Experience preview' tabs. The 'New Web Services Experience preview' tab is selected. Under 'General', there is a 'Published experiment' section with 'View snapshot' and 'View latest' links. A 'Description' section states 'No description provided for this web service.' An 'API key' field contains a long string of characters: 'kPcOtYDwY3waaMRYzstYJsSRVBN+1XapZVsMI+so63DD4bJgAs0fAwlnH1F0ri6CuJ0Y+t...'. Below this, there is a 'Default Endpoint' section with three tabs: 'API HELP PAGE' (selected), 'TEST' (active), and 'APPS'. Under 'TEST', there are two buttons: 'Test' and 'Test preview'. The 'Test preview' button has an icon indicating compatibility with 'Excel 2013 or later'. There are also 'REQUEST/RESPONSE' and 'BATCH EXECUTION' sections with their own 'Test preview' buttons, each with a similar compatibility icon.

We will fill out the data to the experiment



The screenshot shows a 'Test Experiment created on 4/12/2017 [Predictive Exp.] Service' dialog box titled 'Enter data to predict'. It contains five input fields: 'LOAN_AMNT' (value: 0), 'LAST_MEANFICO' (value: 0), 'DTI' (value: 0), 'ADDR_STATE' (empty), and 'EMP_LENGTH' (value: 0). A checkmark icon is located at the bottom right of the dialog.

The output of this experiment is JSON format, which mentions different clusters.



10. Regression

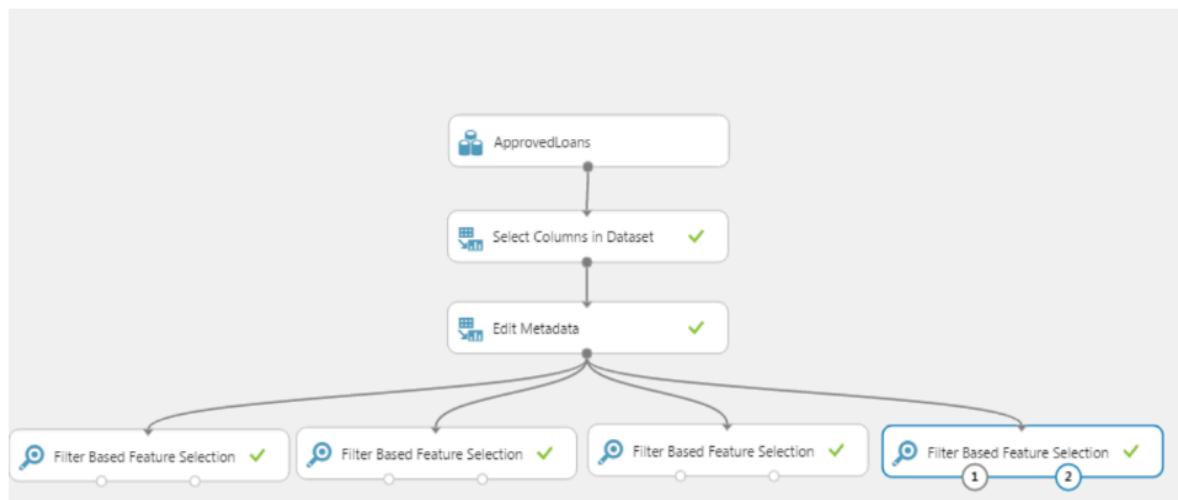
Initial Feature Selection, Categorical Casting, split data, Train Model, Score Model, Evaluate Model are steps in all the models. We will go through the steps as follows.

INITIAL FEATURE SELECTION EXPERIMENT

Since we have 105 features in our cleaned file, not every feature in its current form is expected to contain predictive value to the model, and may mislead or add noise to the model. Some low quality features were removed to improve the model's performance. Low quality includes lack of representative categories, too many missing values, or noisy features.

1. We start the experiment by dragging the dataset.
2. Begin by identifying columns that add little-to-no value for predictive modeling. These columns will be dropped. The columns which will be input to the model.

We select the column names from the original dataset by using various feature selection technique:



Based on the four-feature selection algorithm we came up with 21 columns with major significance on predicting interest rate:

```
'term','purpose','dti','loan_amnt','annual_inc','home_ownership','addr_state','fico_range_high','emp_length','application_type','verification_status','revol_util','inq_last_6mths','open_acc_6m','pub_rec','pub_rec_bankruptcies','delinq_2yrs','open_acc','total_acc','mths_since_last_delinq','mths_since_last_major_derog'
```

3. Categorical casting

Nominal categorical features were identified and cast to categorical data types using the meta data editor to ensure proper mathematical treatment by the machine learning algorithm. To cast these columns, drag in the metadata editor. Specify the columns to be cast, then change the "Categorical" parameter to "Make categorical".

With this, we also start working on the Jupiter notebook to get the best variables present in the cluster.

Performed Variable selection using RFEElimination per segment/cluster. We executed parallel processing for calculating score per cluster.

```
p1 = Process(target=RFELimitation1, args=(X_train1,train1_y,X_test1,test1_y,return_score_p1,"cluster0"))
p2 = Process(target=RFELimitation1,args=(X_train2,train2_y,X_test2,test2_y,return_score_p1,"cluster1"))
p3 = Process(target=RFELimitation1,args=(X_train3,train3_y,X_test3,test3_y,return_score_p1,"cluster2"))
p4 = Process(target=RFELimitation1,args=(X_train4,train4_y,X_test4,test4_y,return_score_p1,"cluster3"))
p5 = Process(target=RFELimitation1,args=(X_train5,train5_y,X_test5,test5_y,return_score_p1,"cluster4"))
p6=_Process(target=KNNAnalysis,args=(X_trainknn,train_y,X_testknn,test_y))

p1.start()
p2.start()
p3.start()
p4.start()
p5.start()
p6.start()

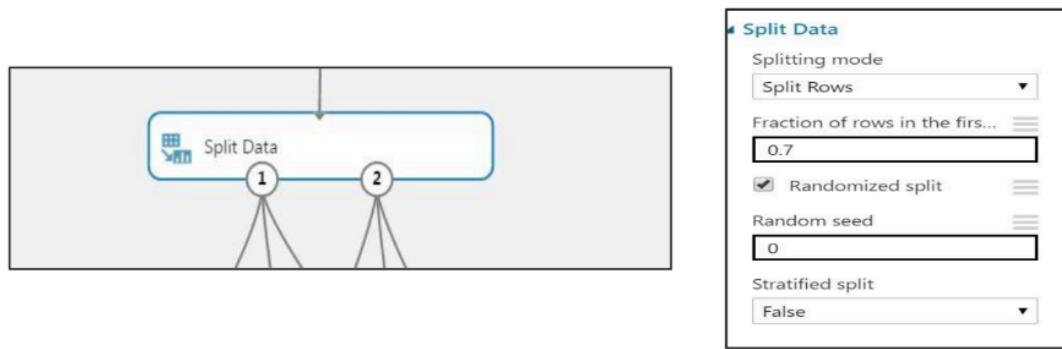
p1.join()
p2.join()
p3.join()
p4.join()
p5.join()
p6.join()

print(return_score_p1)
```

	A	B	C	D	E	F
1	Algorithm Clustering			MAE	RMSE	
2	Cluster 0			0.837366	0.974882	
3						
4	Cluster1			1.592817	1.980042	
5						
6	Cluster2			0.798673	0.998967	
7						
8	Cluster3			0.979527	1.167757	
9						
10	Cluster4			0.817022	0.979817	
11						
12	No Clustering					
13						
14						
15	default			2.610158	3.294504	
16						
17						
18						
19						
20	Manual Clustering					
21						
22	High Fico 60			2.706886	3.502691	
23						
24	High Fico 36			2.199869	2.805017	
25						
26	Low Fico 60			3.035043	3.81311	
27						
28	Low Fico 36			2.694365	3.363686	
29						
30	Medium Fico 60			3.01437	3.817547	
31						
32	Medium Fico 36			2.571373	3.24373	
33						
34						

SPLIT THE DATA

It is extremely important to randomly partition your data prior to training an algorithm to test the validity and performance of your model. A predictive model is worthless to us if it can only accurately predict known values. Withhold data represents data that the model never saw when it was training its algorithm. This will allow you to score the performance of your model later to evaluate how well the model can predict future or unknown values. Randomly split and partition the data into 70% training and 30% scoring using the split module. Drag in a “Split” module. It is usually industry practice to set a 70/30.



Now that the data is ready, constructing a predictive model consists of training and testing. We'll use our data to train the model, and then we'll test the model to see how closely it's able to predict.

LINEAR REGRESSION

Regression is a machine learning used to predict a numeric outcome. Linear regression attempts to establish a linear relationship between independent variables and an outcome variable, or *dependent variable*, which is also numeric.

NEURAL NETWORK REGRESSION

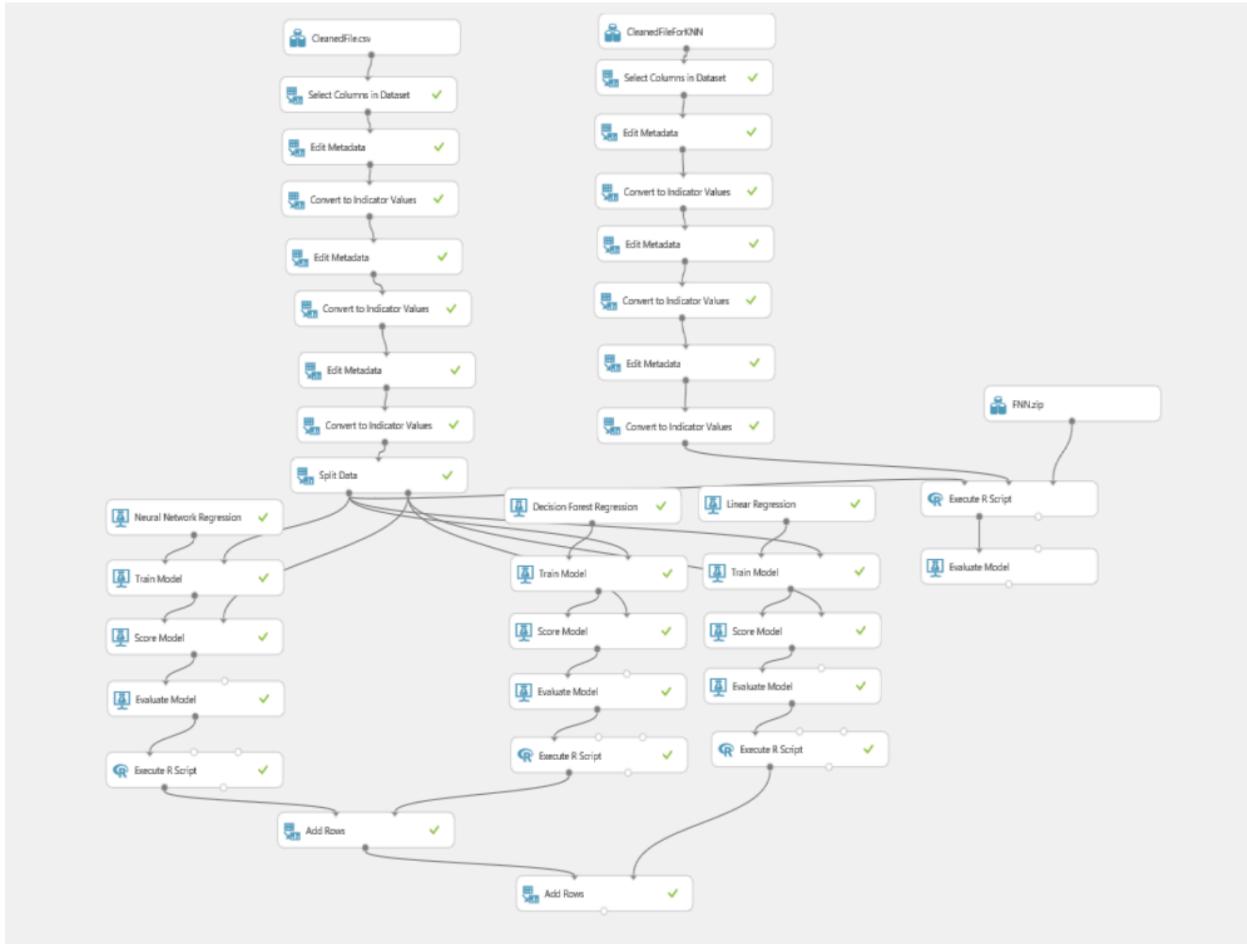
Neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus neural network regression is suited to problems where a more traditional regression model cannot fit a solution.

RANDOM FOREST REGRESSION

Decision trees are non-parametric models that perform a sequence of simple tests for each instance, traversing a binary tree data structure until a leaf node (decision) is reached.

KNN REGRESSION

There is no in-built component for KNN regression provided by Azure ML studio so we have created a function in Jupyter notebook to check the accuracy of KNN regression model



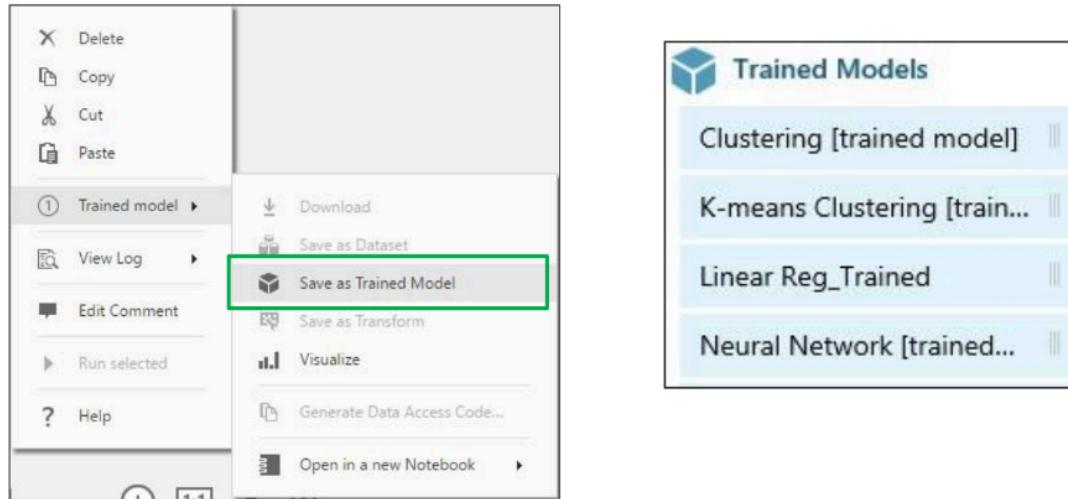
Assignment2-Prediction Algorithms > Add Rows > Results dataset

rows	columns	Algorithm	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error	Relative Squared Error	Coefficient of Determination
3	6	view as					
		Neural Network	0.145084	0.298205	0.040049	0.004257	0.995743
		Decision Forest	1.292151	2.045128	0.356686	0.200241	0.799759
		Linear Regression	2.229361	2.850165	0.615394	0.388913	0.611087

We choose Neural Network for Predicting interest rate as its Root Mean Error was less.

SAVING TRAINED MODEL

After we run all the models and is satisfied with the Scored Model output, we proceed to save the model by right clicking it.



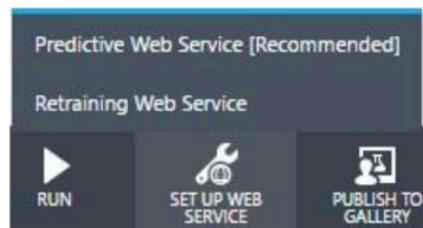
SETUP WEB SERVICE: REGRESSION

CONVERT THE TRAINING EXPERIMENT TO A PREDICTIVE EXPERIMENT

Converting to a predictive experiment involves three steps:

1. Save the model we've trained and then replace our training modules
2. Trim the experiment to remove modules that were only needed for training
3. Define where the Web service will accept input and where it generates the output

All three steps can be accomplished by clicking "Set Up Web service" at the bottom of the experiment.

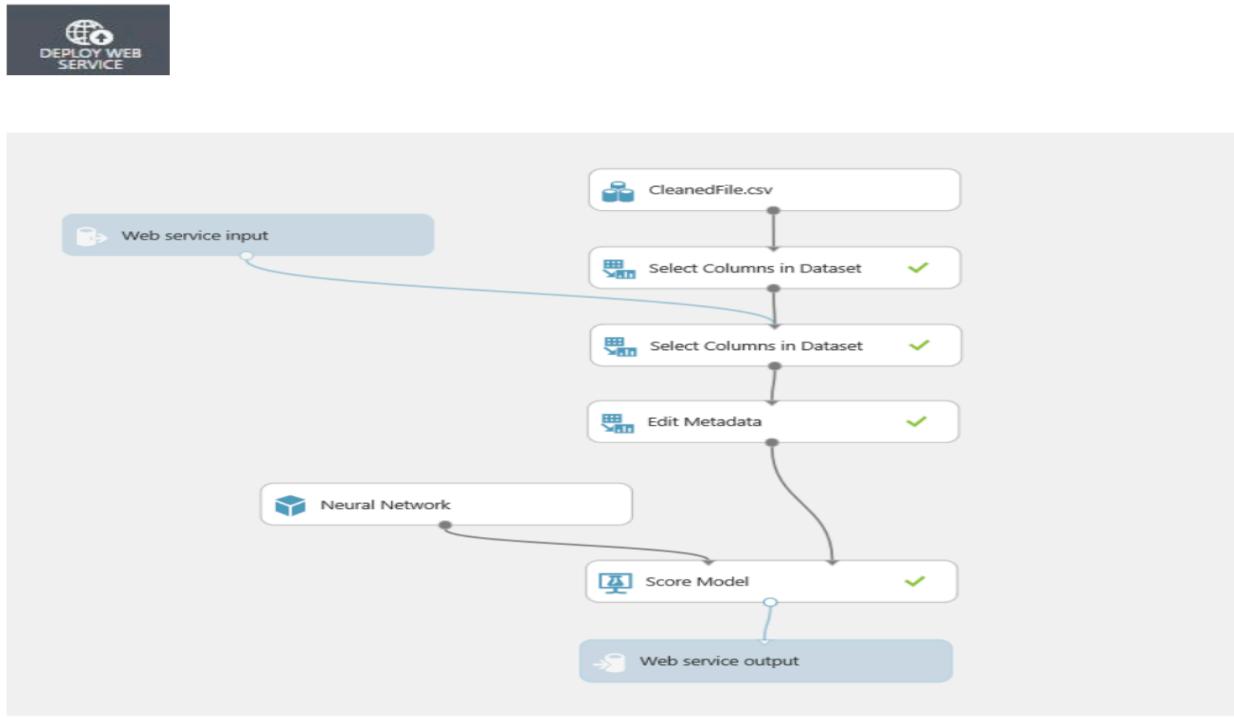


When we click Set Up Web service, several things happen:

The trained model is saved as a single Trained Model module into the module palette to the left of the experiment canvas (we can find it under Trained Models). Modules that were used for training are removed.

DEPLOY THE WEB SERVICE

We can deploy the experiment as either a classic Web service or a new Web service based on Azure Resource Manager.



To deploy a classic Web service derived from our experiment, click **Deploy Web Service** below the canvas and select **Deploy Web Service [Classic]**. Machine Learning Studio deploys the experiment as a Web service and takes you to the dashboard for that Web service.

From here, you can return to the experiment (**View snapshot** or **View latest**) and run a simple test of the Web service (See **Test the Web service** below).

The screenshot shows a user interface for predicting data. The title bar reads 'Test Experiment created on 4/12/2017 [Predictive Exp.] Service'. The main area is titled 'Enter data to predict' and contains five input fields:

- LOAN_AMNT: A text input field containing '0'.
- LAST_MEANFICO: A text input field containing '0'.
- DTI: A text input field containing '0'.
- ADDR_STATE: An empty text input field.
- EMP_LENGTH: A text input field containing '0'.

A vertical scroll bar is visible on the right side of the form.

The results will be a JSON format output.

WEB IMPLEMENTATION

We will be using Angular.js as our frontend framework to call the Rest API for each part.

Angular.js has following main component's

(1) APP.JS

We will define out Application name as shown below

```
var myApp = angular.module("productCustomizer", ["ngRoute", "rzModule"]);
```

(2) CONFIG.JS

A Configuration file will need to be made for the routing of the web page. We will define our main page as index.html and a controller which is required by Angular as Main Controller.

```
module('productCustomizer').config(Config);  
function Config($routeProvider){  
    $routeProvider  
        .when('/', {  
            templateUrl: 'index.html',  
            controller: 'MainController'  
        })  
        .otherwise({  
            redirectTo: '/'  
        });  
}());
```

This will be default location for the server control. Application port's, redirection and function's to be called for REST api will be handled here

First, we will load all the required libraries for the HTTP functionality as shown below

```

1 var express = require('express');
2 var app = express();
3 var http = require("http");
4
5 var https = require("https");
6
7 var querystring = require("querystring");
8
9 var fs = require('fs');
10
11 var bodyParser = require('body-parser');
12 app.use(bodyParser.json());
13 app.use(bodyParser.urlencoded({ extended: true }));
14
15 // configure a public directory to host static content
16 app.use(express.static(__dirname + '/public'));
17
18
19 var ipaddress = process.env.OPENSHIFT_NODEJS_IP;
20 var port      = process.env.OPENSHIFT_NODEJS_PORT || 3000;
21

```

Then, we will define our application's posting url. This is essentially the routing post, where based on the type of prediction we will create different functions

```

// app.post('/prediction/linear-regression', linearRegression);
app.post("/clustering", clustering);
app.post("/prediction/regression", regression);
app.post("/classification", classification);
app.post("/clustering/zero", clusteringzero);
app.post("/clustering/one", clusteringone);
app.post("/clustering/two", clusteringtwo);
app.post("/clustering/three", clusteringthree);
app.post("/clustering/four", clusteringfour);

```

So we will use three different post for three different type of predictions that we will perform

Now we can define our getPred function which we will use to call our api call and get the result back. This function will take data of the user input, api key and url for the call to work.

(3) SERVER.JS

```

function getPred(data, path, api_key) {
  var dataString = JSON.stringify(data);
  var host = 'ussouthcentral.services.azureml.net';
  var headers = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + api_key};

  var options = {
    host: host,
    port: 443,
    path: path,
    method: 'POST',
    headers: headers
  };

  result = '';
  var reqPost = [https].request(options, function (res) {
    res.on('data', function (d) {
      setTimeout(function() {
        process.stdout.write(d);
        result += d;
      }, 300);
    });
  });

  // Would need more parsing out of prediction from the result
  reqPost.write(dataString);
  reqPost.end();
  reqPost.on('error', function (e) {
    console.error(e);
  });
  console.log(result);
  //return result;
}

```

Output of the function will be stored in a result variable and can be consumed later

Now since we have three different calls to make, we will define 3 different functions which will basically provide respective data, api and url to the getPred function defined above as shown below

```

function classification(req, res){
  var data = req.body;
  //console.log(data);
  var path = '/workspaces/cb863d4f3f39452ba855f979fdc9c60c/services/58acc102ceff417bb446bc236559a2ce/';
  var key = 'r2AAk9wLsUH6MGBaxUdTtqN8ZDlNwr8is2kHNcMKH0Jk8UD+JQPCBFv8qomu/rrDPav025LLr+BVlb9Y+fjHjg==';
  getPred(data, path, key);
  setTimeout(function() {
    //console.log(result);
    res.json(result);
  }, 1000);
}

function regression(req, res){
  var data = req.body;
  console.log(data);
  var path = '/workspaces/5f59fa9fa20e40bdbb68b87d4b1e7006/services/a6b2e4a5b11f4a7699929777cc4a4e81/';
  var key = 'hwWT5Ubnu3QYJGKaWFvpQqGKRHerg+SUYJ+8NxXULPfE07xZ9lHyIov0SA2bxP7GM/G4hC+m+m1Tb2VWbTyTg==';
  getPred(data, path, key)

  //console.log(result);
  setTimeout(function() {
    res.json(result);
  }, 1000);
}

function clustering(req, res){
  var data = req.body;
  //console.log(data);
  var path = '/workspaces/895503c96ee3453e8e49e4a6b911d739/services/3a81368bbb374858974992c2a30dfaee/';
  var key = 'XbE69XAEME3HPkChQP4PEPdBGeF/o5yC16h9kG0T7L3PyEnvfmwx7Y/SggA8oehC36nYzifgf4Cf10JKP6zeWg==';
  getPred(data, path, key)
}

```

Now before we jump to MainController, we will design our web page. Since, MainController require elements fetch from the front end, we will define the home page and use those element's in the MainController later

(4) INDEX.HTML

We will say the home page to use our app defined by ng-app. in the main html tag. Similarly, we will add scripting components to use required angular.js libraries.

We will provide ng-controller “Main Controller” to the main body section, which we will define later. We will also add custom script’s we defined earlier (app.js,config.js and main.controller.js)

```

1 <!DOCTYPE html>
2 <html lang="en" ng-app="productCustomizer">
3 <head>
4   <title> Assignment 2 – ADS – Spring 2017 </title>
5   <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.0/angular.js"></script>
6   <script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.5.0-beta.0/angular-route.js"></script>
7   <link rel="stylesheet" type="text/css" href="//fonts.googleapis.com/css?family=Open+Sans" />
8 </head>
9 <body ng-controller="MainController" style="padding: 20px;">
0 <style>
1   .high{
2     background-color: lightgreen;
3   }
4 </style>
5 <h1>Lending Club Loan Portal</h1>
6 <h3> An easy way to get loans, check your interest now!!</h3>
7 <br/><br/>
8 <script src="app.js"></script>
9 <script src="config.js"></script>
0 <script src="main.controller.js"></script>
1

```

For the user control on the type of Prediction to be used, will provide 3 button’s and link those button’s to ng-method’s to distinguish and control the flow.

For the type of features we decided for each type of Prediction, we will use div components and tag them with related ng-method. For example, if we want to use Temperature as a component for Regression, we will mark the div component as ng-method == “Prediction” which we defined earlier.

so on and so forth respectively for each type of Prediction method. We can see the result of the HTML page below:

Classification:

Loan Amount	10000
Credit Score (FICO)	692
Debt-to-Income Ratio (DTI)	20
State	CA
Employment Length	10
Purpose	other

Output:

Status	Probablity
Approved	0.67733674579017

Interest Ratesz:

Loan Amount	10000
Credit Score (FICO)	692
Debt-to-Income Ratio (DTI)	20
State	CA
Employment Length	10
Purpose	other
Term	36
Home Ownership	RENT
Annual Income (in dollars)	231321312
Verification Status	Verified
Number of Delinquency (in last 2 years)	0
Inquiry in last 6 months	12
Months since last delinquent	0
Open Accounts	12

Algotithm Clustering Output:

Algorithm Used	Assigned Cluster	Calculating Interest Rates
KMeans-clustering	1	16.7006985525391

Default Interest : 6.86438274383545