Advance Data Science

LOAN LENDING CLUB

Assignment – 3 Part - 1

Report by:

Ankit Bhayani

Rajat Agrawal

Vaisakha Sawant

OBJECTIVE

The report summarizes the design and implementation of the data wrangling performed on the Loan Lending club data set. This report is divided into three section.

Section 1: Describes the step by step implementation of downloading the dataset provided on the website by the Loan Lending club and preprocessing the data.

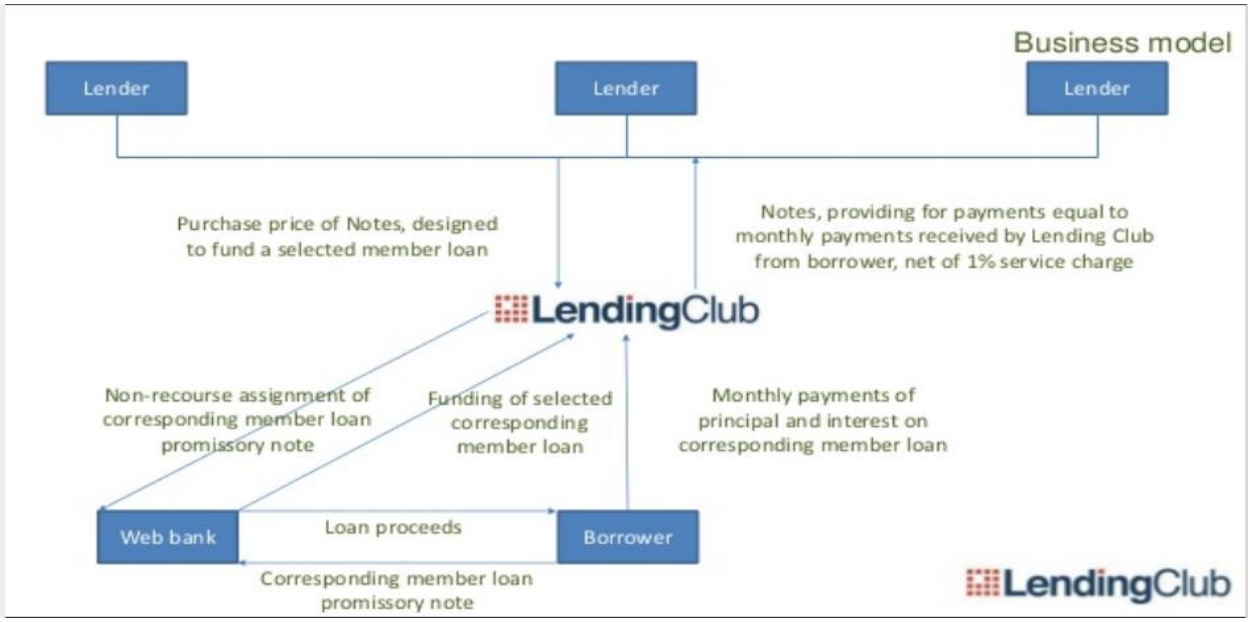
Section 2: Describes the step by step process to clean the downloaded file and apply feature engineering technique to predict interest rate.

Section 3: Describe the implementation to streamline the process using Luigi which will run inside a Docker and host the resultant output on Amazon S3.

SECTION - 1 Data Download & Pre-Processing:

OVERVIEW

Lending Club is the world's largest online credit marketplace for peer to peer lending, facilitating personal loans, business loans, and financing for elective medical procedures. Borrowers access these loans through fast and easy online or mobile interfaces. Investors provide the capital to enable many of the loans in exchange for earning interest. Peer-to-peer (P2P) lending is the practice of lending money to individuals or businesses through online services that match lenders directly with borrowers. Since the P2P lending companies offering these services operate entirely online, they can run with lower overhead and provide the service more cheaply than traditional financial institutions. Thus, lenders often earn higher returns compared to savings and investment products offered by banks, while borrowers can borrow money at lower interest rates, even after the P2P lending company has taken a fee for providing the match-making platform and credit checking the borrower. The interest rates are set by lenders who compete for the lowest rate on the reverse auction model or fixed by the intermediary company based on an analysis of the borrower's credit. Fig. 1.



Lending Club's Business Model: The process of an application for a loan to its approval can be explained primarily in three steps:

- Customers interested in a loan complete a simple application at LendingClub.com.

- Lending Club leverages online data and technology to quickly assess risk, determine a credit rating and assign appropriate interest rates. Qualified applicants receive offers and can evaluate loan options with no impact to their credit score.

- Investors ranging from individuals to institutions select loans in which to invest and can earn monthly returns.

STEP 1:

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.

```
def getData():
    url='https://www.lendingclub.com/account/login.action?'
    postUrl='https://www.lendingclub.com/info/download-data.action'
    payload={'login_email':'agrawal.r@husky.neu.edu','login_password':'ADS@12345'}
    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" : 'loanStatsFileNamesJS' })
        Borrowerdata=[]
        Loandata=[]
        loanyearlist=[]
        for div in ziplist:
            for d in div :
                link = d.split('|')
                Borrowerdata.extend(link)
        for data in Borrowerdata:
            if data !='':
                Loandata.append('https://resources.lendingclub.com/'+ data)
        year_options = soup.findAll('select',{"id":'loanStatsDropdown'})[0].findAll("option")
        for year in year_options:
            loanyearlist.append(year.text)
        dictionary = dict(zip(loanyearlist, Loandata))


        for year in dictionary:
            url=dictionary[year]
            yearpath=str(os.getcwd())+"\\"+year
            extracrtZip(url,yearpath)
```

Since we are using python 3.2 for accessing the URL, we should import the following python library to access the Loan Lending club website.

```
import re
import requests
from bs4 import BeautifulSoup
import urllib.request
import csv
import sys
import os
from zipfile import ZipFile
import logging
from io import BytesIO
import pandas as pd
import numpy as np
import glob
from bs4 import BeautifulSoup
from urllib.request import urlopen
from zipfile import ZipFile
from io import BytesIO
```

We are storing the user login information in a request session, which allows us to download the actual files present on the Lending Loan club website.

```
        # consolidate data
        writeHeader2 = True
        for year in dictionary:
            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)
```

```
getData()
```

```
LoanStats_securev1_2016Q4.csv
LoanStats3a_securev1.csv
LoanStats3c_securev1.csv
LoanStats_securev1_2016Q1.csv
LoanStats_securev1_2016Q2.csv
LoanStats_securev1_2016Q3.csv
LoanStats3b_securev1.csv
LoanStats3d_securev1.csv
```

The same method is applied for downloading the file having details of the application which are rejected for granting the loan.

```
def getDeclinedData():
    url='https://www.lendingclub.com/account/login.action?'
    postUrl='https://www.lendingclub.com/info/download-data.action'
    payload={'login_email':'agrawal.r@husky.neu.edu','login_password':'ADS@12345'}
    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))


        for year in dictionary:
            url=dictionary[year]
            yearpath=str(os.getcwd())+"\\RejectData\\"+year
            extracrtZip(url,yearpath)
```

```
        # consolidate data
        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,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)
```

```
getDeclinedData()
```

```
RejectStatsB.csv
RejectStats_2016Q4.csv
RejectStats_2016Q1.csv
RejectStatsD.csv
RejectStatsA.csv
RejectStats_2016Q3.csv
RejectStats_2016Q2.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)
```

As shown, resultant loan data frame consists of 1321848 rows with 115 columns.

Same step was performed on the rejected loans application as shown below:

```
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%
```

After removing insignificant columns from our dataset, we will work on handling missing data and data cleaning.

Since we have large number of columns in our dataset we will first focus on those columns which are significant in predicting loan interest rate.

```
#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_del
#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','annual_inc','delinq_2yrs','fico_range_low','fico_range_high','inq_l
    df[['mths_since_last_delinq','open_acc','pub_rec','revol_bal','total_acc','last_meanfico']] = df[['mths_since_last_d
    df[['int_rate','revol_util']] = df[['int_rate','revol_util']].astype('float64')
    return df
```

Cleaned Loan Data Set:

```
loan_df.head()
```

	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['Employment Length'].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

Once we have all the data in the cleaned form, we can use this data for selecting features that significant for predicting the Loan interest rate. Since in the cleaned data set, we still have 108 columns, we need to determine the columns which are provided by the user who is requesting loan from the lending club and the other significant column which are provided by third party to Lending Loan club to determine whether to provide Loan to the applicant or not.

This can be inferred from the data dictionary provided by the Laon Lending club which consist of all the available columns and their definition.

LCDataDictionary.xlsx

We used standard multivariate linear regression model to associate interest rate with other parameters presented in the data. We selected this model assuming linear relationship of interest rate and FICO range. We also suggested the same kind of relationship between interest rate and some other variables, which we used in our analysis.

```
#Feature Engineering
import warnings
from sklearn.linear_model import (LinearRegression, Ridge,
                                  Lasso, RandomizedLasso)
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")
loanfreature_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 =loanfreature_df.int_rate
loanfreature_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
loanfreature_df = loanfreature_df[loanfreature_df['home_ownership'].isin(home_positive + home_negative)].copy()
loanfreature_df['home_ownership_category']=loanfreature_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']
loanfreature_df = loanfreature_df[loanfreature_df['verification_status'].isin(verification_positive + verification_negat
loanfreature_df['verification_status_category']=loanfreature_df['verification_status'].isin(verification_positive).astyp
```

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']
loanfreature_df = loanfreature_df[cols_to_keep]
loanfreature_df = createDummies(loanfreature_df)
X = loanfreature_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 ))
```

```
#Comapring 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%s" % "\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")
```

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 indicationg 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 creditworthness in the 6 months before the credit was issued (Inquiries.in.the.Last.6.Months)
- Length of time employed at current job (Employment.Length)

FeatureSelection.csv

	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 inquires 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.

```
correction_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 Loan club.

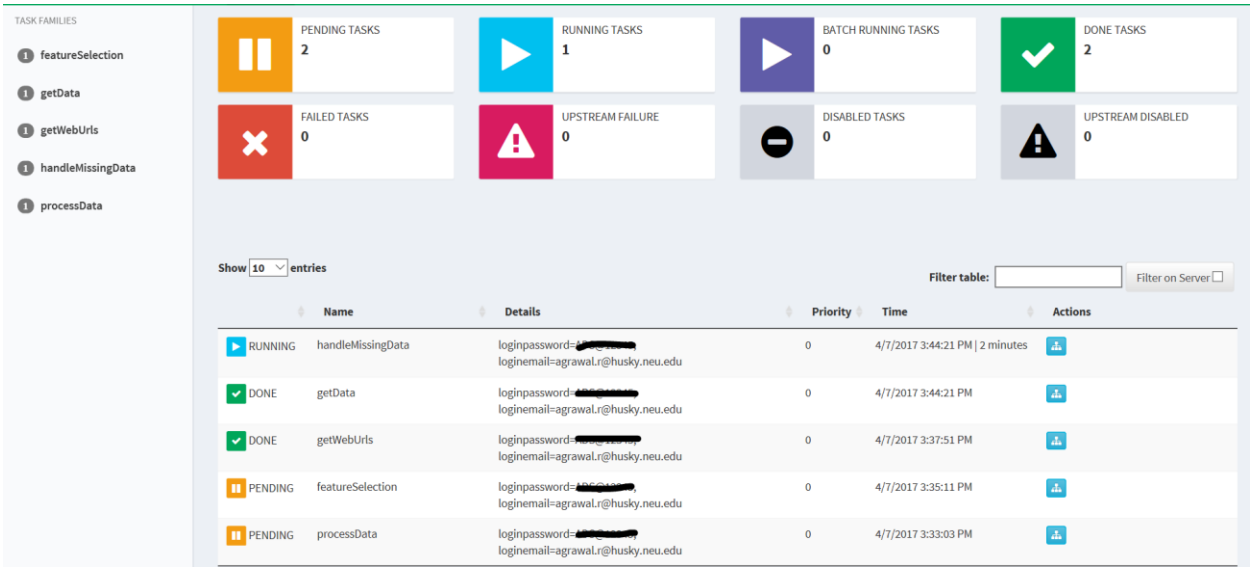
- **Amount Requested**
- **Fico_Score**
- **Annual Income**
- **Employment Length**

SECTION – 3 PIPELINING PROCESS USING LUIGI

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

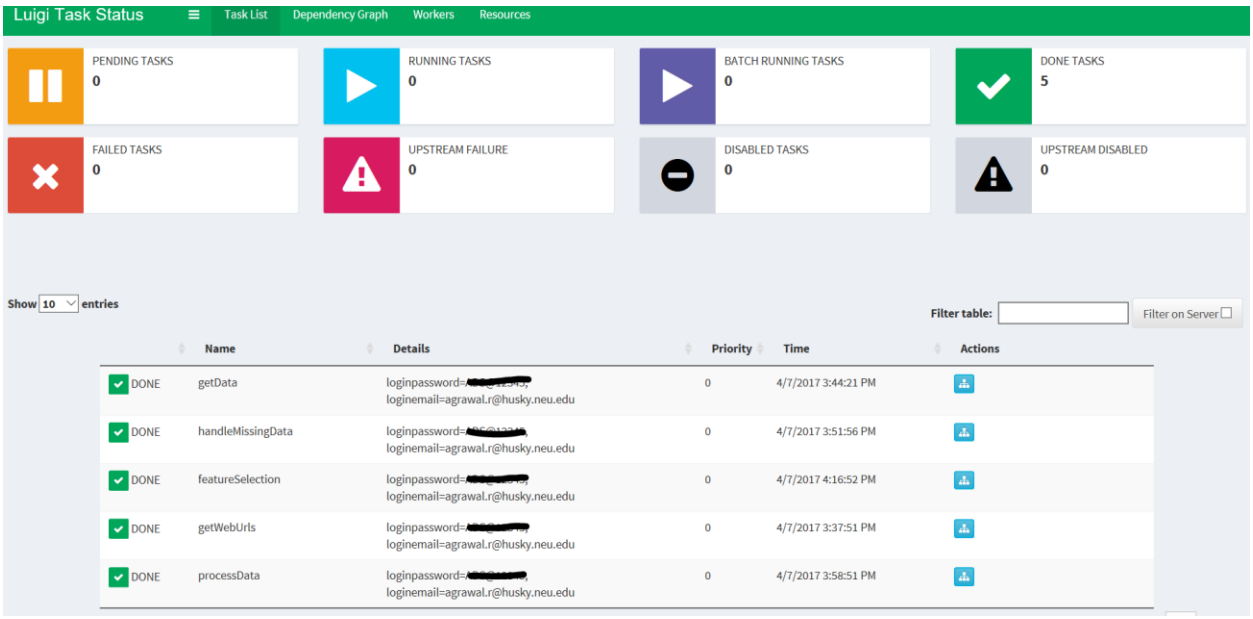
Luigi performs following functions in a pipeline for Loan data which is approved:

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 techniqiues.



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

Time taken by each task can be seen as follows:



Similarly, we implemented Luigi pipeline for rejected data:

DOCKERIZE PART 1

STEP 1:

CREATE THE DOCKER IMAGE FROM DOCKERFILE

1.1 After taking the base UBUNTU 14.04 image.

Dockerfile

```
1
2   FROM ubuntu:14.04
3
4   MAINTAINER Ankit Bhayani <bhayani.a@husky.neu.edu>
5
6   USER root
7
8   # Install dependencies
9   RUN apt-get update && apt-get install -y \
10      python-pip --upgrade python-pip
11
12  RUN pip install --upgrade pip
```

1.2 Install Python and all required packages which is used in source code

```
# install py3
RUN apt-get update -qq \
 && apt-get install --no-install-recommends -y \
    # install python 3
    python3 \
    python3-dev \
    python3-pip \
    python3-setuptools \
    pkg-config \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

RUN pip3 install --upgrade pip

# install additional python packages
RUN pip3 install ipython
#RUN pip install jupyter
RUN pip3 install numpy
RUN pip3 install pandas
RUN pip3 install scikit-learn
RUN pip3 install BeautifulSoup4
RUN pip3 install scipy
#RUN pip install nltk

#install AWS CLI
RUN pip3 install awscli
```

1.3 While building Docker File; First creating the work directory and then copying the source python and shell files from local machine to DOCKER image.

```
WORKDIR /src
RUN mkdir /src/Output

ADD PART1-driver.sh /src
ADD PART1-downloader_luigi.py /src
ADD PART1-declined_luigi.py /src
ADD awsS3Upload.sh /src
RUN chmod +x /src/PART1-driver.sh
RUN chmod +x /src/awsS3Upload.sh
```

STEP 2:

BUILD THE DOCKERFILE TO CREATE THE DOCKER IMAGE

Building this Docker file will create a DOCKER image with name ankitbhayani/FirstAssignmentImage

```
ANKITs-MacBook-Pro:dockerImage ankitbhayani$ docker build -t ankitbhayani/assignment2part1 .
```

Each step in this file will be executed in order to create an image. AWS CLI is also integrated inside the dockerfile

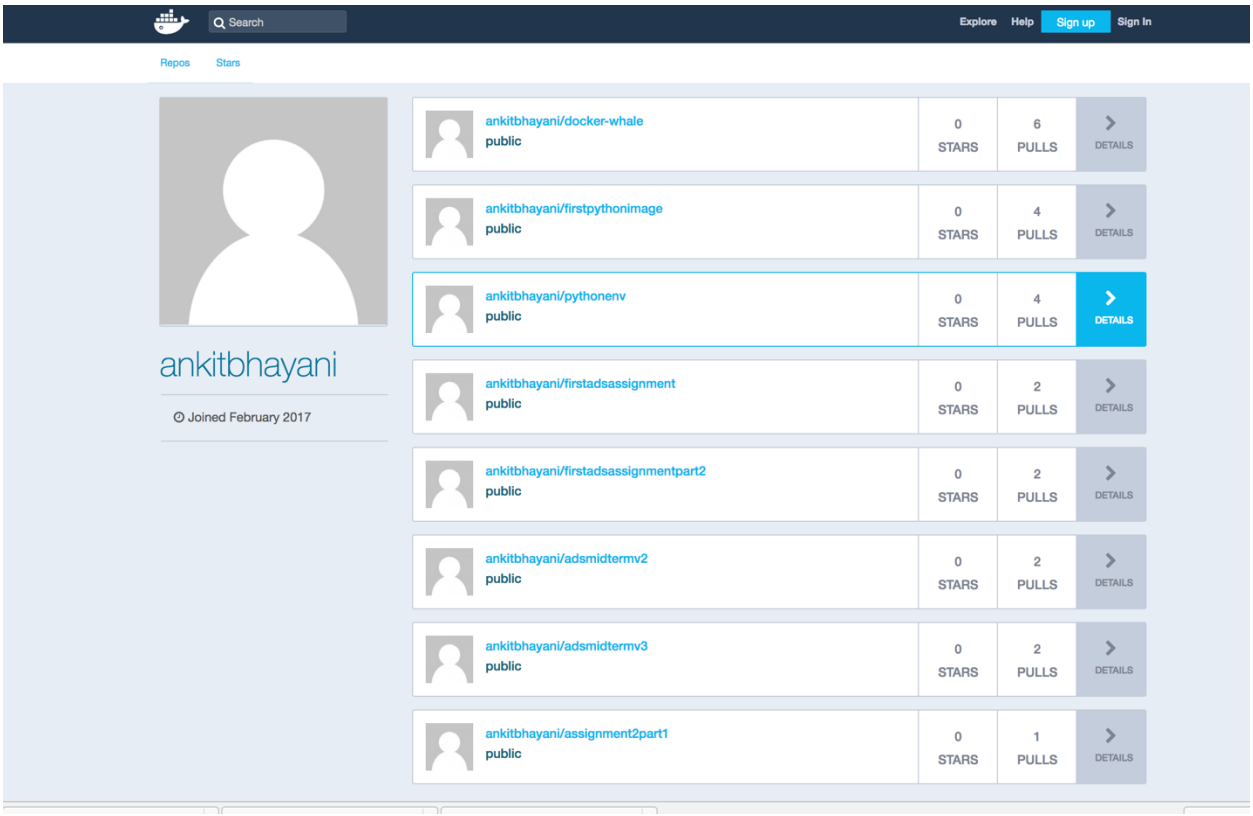
```
ANKITs-MacBook-Pro:FirstAssgnmntDockerFile ankitbhayani$ docker build -t ankitbhayani/firstadsassignment .
Sending build context to Docker daemon 433.6 MB
Step 1/26 : FROM ubuntu:14.04
 ---> b969ab9f929b
Step 2/26 : MAINTAINER Ankit Bhayani <bhayani.a@husky.neu.edu>
 ---> Using cache
 ---> 48e95dea2a76
Step 3/26 : USER root
 ---> Using cache
 ---> 36c635875807
Step 4/26 : RUN apt-get update && apt-get install -y     python-pip --upgrade python-pip
 ---> Using cache
 ---> 369a3ef895a9
Step 5/26 : RUN pip install --upgrade pip
 ---> Using cache
 ---> cbb2916ae26d
Step 6/26 : RUN apt-get update -qq  && apt-get install --no-install-recommends -y     python3     python3-dev     python3-pip     python3-setuptools     pkg-config  && apt-get clean
  && rm -rf /var/lib/apt/lists/*
 ---> Using cache
 ---> 13b83f46b37c
Step 7/26 : RUN pip3 install --upgrade pip
 ---> Using cache
 ---> 69d0e36292c9
Step 8/26 : RUN pip3 install ipython
 ---> Using cache
 ---> cb457088d5be
Step 9/26 : RUN pip3 install numpy
 ---> Using cache
 ---> 4d96a3f9156d
Step 10/26 : RUN pip3 install pandas
 ---> Using cache
 ---> e37f80191dac
Step 11/26 : RUN pip3 install scikit-learn
 ---> Using cache
 ---> 9df957442ebe
Step 12/26 : RUN pip3 install BeautifulSoup4
 ---> Using cache
 ---> 5058090e4828
Step 13/26 : RUN pip3 install scipy
 ---> Using cache
 ---> 16459c9781f7
Step 14/26 : RUN pip3 install awscli
 ---> Using cache
 ---> 144cb985eef1
Step 15/26 : RUN echo 'alias ll="ls --color=auto -lA"' >> /root/.bashrc  && echo '"\e[5~": history-search-backward' >> /root/.inputrc  && echo '"\e[6~": history-search-forward' >> /
root/.inputrc
 ---> Using cache
 ---> f1bc7822f760
Step 16/26 : ENV PASSWD 'sha1:98b767162d34:8da1bc3c75a0f29145769edc977375a373407824'
 ---> Using cache
 ---> b858b8dbc1cf
Step 17/26 : RUN dpkg-query -l > /dpkg-query-l.txt  && pip2 freeze > /pip2-freeze.txt  && pip3 freeze > /pip3-freeze.txt
 ---> Using cache
 ---> 69e55017a8bc
Step 18/26 : EXPOSE 8888
 ---> Using cache
 ---> 82c5106f4b40
Step 19/26 : WORKDIR /src/
```

```
 ---> d5f6caa394f7
Step 18/34 : RUN pip3 install git+https://github.com/pybrain/pybrain.git@0.3.3
 ---> Using cache
 ---> 1ef77b16e493
Step 19/34 : RUN pip3 install awscli
 ---> Using cache
 ---> de4866d4bb12
Step 20/34 : RUN pip3 install luigi
 ---> Using cache
 ---> 2ed704dd47d5
Step 21/34 : RUN echo 'alias ll="ls --color=auto -lA"' >> /root/.bashrc  && echo '"\e[5~": history-search-backward' >> /root/.inputrc  && echo '"\e[6~": history-search-forw
 ---> Using cache
 ---> a496e9ee7aed
Step 22/34 : ENV PASSWD 'sha1:98b767162d34:8da1bc3c75a0f29145769edc977375a373407824'
 ---> Using cache
 ---> 8c71d338e007
Step 23/34 : RUN dpkg-query -l > /dpkg-query-l.txt  && pip2 freeze > /pip2-freeze.txt  && pip3 freeze > /pip3-freeze.txt
 ---> Using cache
 ---> 36c30384f7ea
Step 24/34 : EXPOSE 8888
 ---> Using cache
 ---> 89c81c8f335b
Step 25/34 : RUN mkdir -p -m 700 /root/.jupyter/ &&     echo "c.NotebookApp.ip = '*'" >> /root/.jupyter/summary_part1.ipynb
 ---> Using cache
 ---> e055164897a4
Step 26/34 : WORKDIR /src
 ---> Using cache
 ---> f1d5e41f5102
Step 27/34 : RUN mkdir /src/Output
 ---> Running in 3968e2e5566f
 ---> 0793bccb1bac
Removing intermediate container 3968e2e5566f
Step 28/34 : ADD PART1-driver.sh /src
 ---> aa8bd1c25edb
Removing intermediate container 9dcaba28df53
Step 29/34 : ADD PART1-downloader_luigi.py /src
 ---> 43b5dbe86277
Removing intermediate container 3ba957c9829d
Step 30/34 : ADD PART1-declined_luigi.py /src
 ---> bb63442c74b4
Removing intermediate container 5e5680f63d9f
Step 31/34 : ADD awsS3Upload.sh /src
 ---> 20d7f064d869
Removing intermediate container d1a2592b66e9
Step 32/34 : RUN chmod +x /src/PART1-driver.sh
 ---> Running in ba902f31139f
 ---> 7435efca13d2
Removing intermediate container ba902f31139f
Step 33/34 : RUN chmod +x /src/awsS3Upload.sh
 ---> Running in 5c8a2f25a96f
 ---> df7d5168cd3a
Removing intermediate container 5c8a2f25a96f
Step 34/34 : CMD /bin/bash -c 'jupyter notebook --no-browser --ip=* --NotebookApp.password="$PASSWD" "$@"'
 ---> Running in 1a0f489dcc55
 ---> 858002959291
Removing intermediate container 1a0f489dcc55
Successfully built 858002959291
```

STEP 3:

PUSH THIS DOCKER IMAGE to DOCKERHUB

```
ANKITs-MacBook-Pro:dockerImage ankitbhayani$ docker push ankitbhayani/assignment2part1
The push refers to a repository [docker.io/ankitbhayani/assignment2part1]
d05fbe947ede: Pushed
fec6327c656d: Pushed
6905946d443a: Pushed
de5d834a122a: Pushed
a76c2a06dca3: Pushed
03feff07b197: Pushed
f7203bdcf2c2: Pushed
bf2fe17b34a3: Pushed
61e733c4db90: Pushed
1d1f08f44b1a: Pushed
1027dc82c3dd: Pushed
87c5f20175d1: Pushed
11a1b64d078b: Pushed
18c617de41b2: Mounted from ankitbhayani/adsmidtermv3
d53019374e7a: Mounted from ankitbhayani/adsmidtermv3
782e87023c86: Mounted from ankitbhayani/adsmidtermv3
48fa3c1db85e: Mounted from ankitbhayani/adsmidtermv3
27eb859c2e6c: Mounted from ankitbhayani/adsmidtermv3
5777720d922c: Mounted from ankitbhayani/adsmidtermv3
b29af706936f: Mounted from ankitbhayani/adsmidtermv3
45302c372334: Mounted from ankitbhayani/adsmidtermv3
f8b852f7c85f: Mounted from ankitbhayani/adsmidtermv3
a10d1e7c5cee: Mounted from ankitbhayani/adsmidtermv3
a93537600a1c: Mounted from ankitbhayani/adsmidtermv3
3f2932e47bae: Mounted from ankitbhayani/adsmidtermv3
a4f935039205: Mounted from ankitbhayani/adsmidtermv3
c12bac02a4bf: Mounted from ankitbhayani/adsmidtermv3
84a834218e58: Mounted from ankitbhayani/adsmidtermv3
bd00cdbae641: Mounted from ankitbhayani/adsmidtermv3
af43131c4039: Mounted from ankitbhayani/adsmidtermv3
9bd4c7af882a: Mounted from ankitbhayani/adsmidtermv3
04ab82f865cf: Mounted from ankitbhayani/adsmidtermv3
c29b5eadf94a: Mounted from ankitbhayani/adsmidtermv3
```



STEP 4:

DOCKER RUN INSTRUCTIONS:

Pull an image from dockerhub:

docker pull ankitbhayani/assignment2part1

Run the docker image (for Lending-Club full data):

docker run -e ACCESS_KEY=**<YOUR_ACCESS_KEY>**

-e SECRET_KEY=**<YOUR_SECRET_KEY>**

-e S3_PATH=**<YOUR_S3_PATH>**

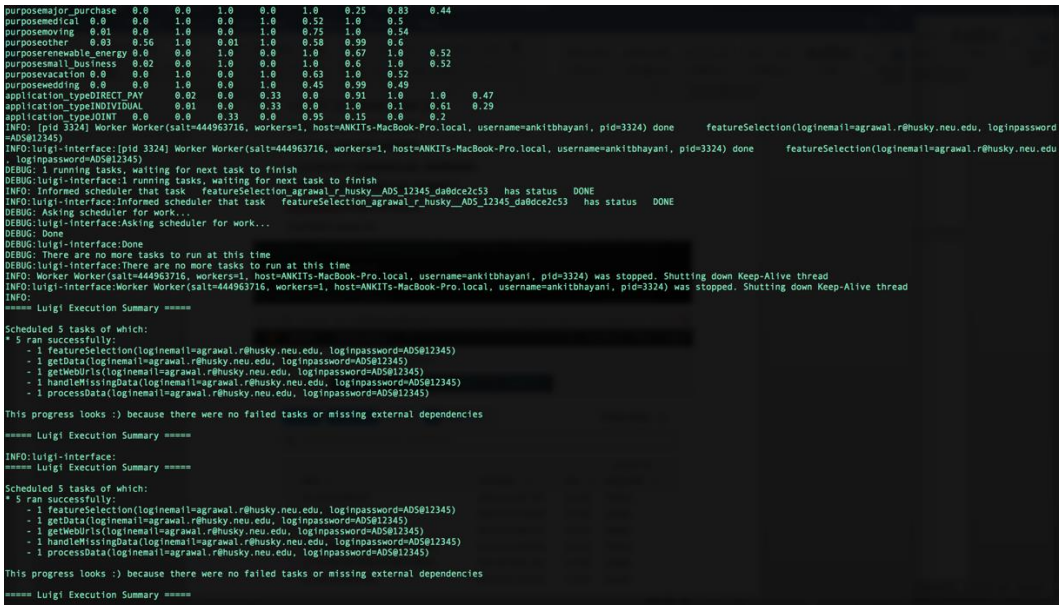
-e REGION=**<BUCKET_REGION>**

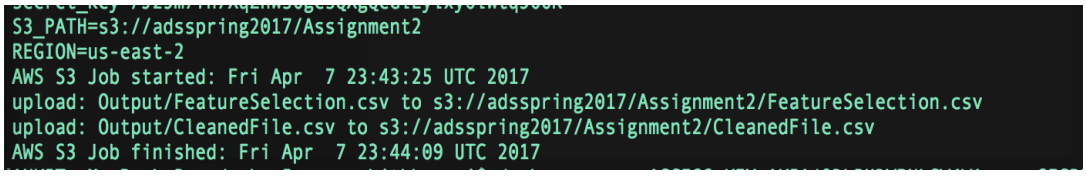
-e USERNAME=**<LENDINGCLUB_USERNAME>**

-e PASSWD=**< LENDINGCLUB_PASSWD >**

ankitbhayani/assignment2part1

/src/PART1-driver.sh





Run the docker image (for Lending-Club Declined data):

docker run -e ACCESS_KEY=**<YOUR_ACCESS_KEY>**

-e SECRET_KEY=**<YOUR_SECRET_KEY>**

-e S3_PATH=**<YOUR_S3_PATH>**

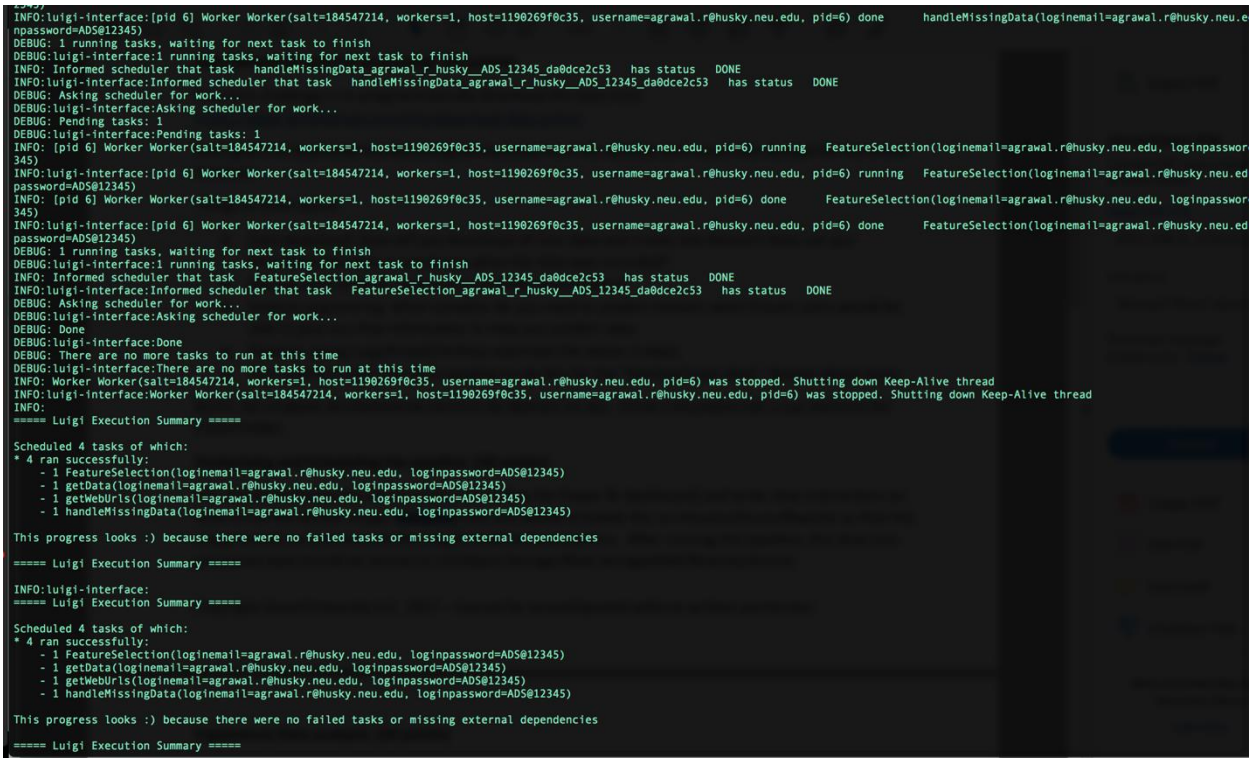
-e REGION=**<BUCKET_REGION>**

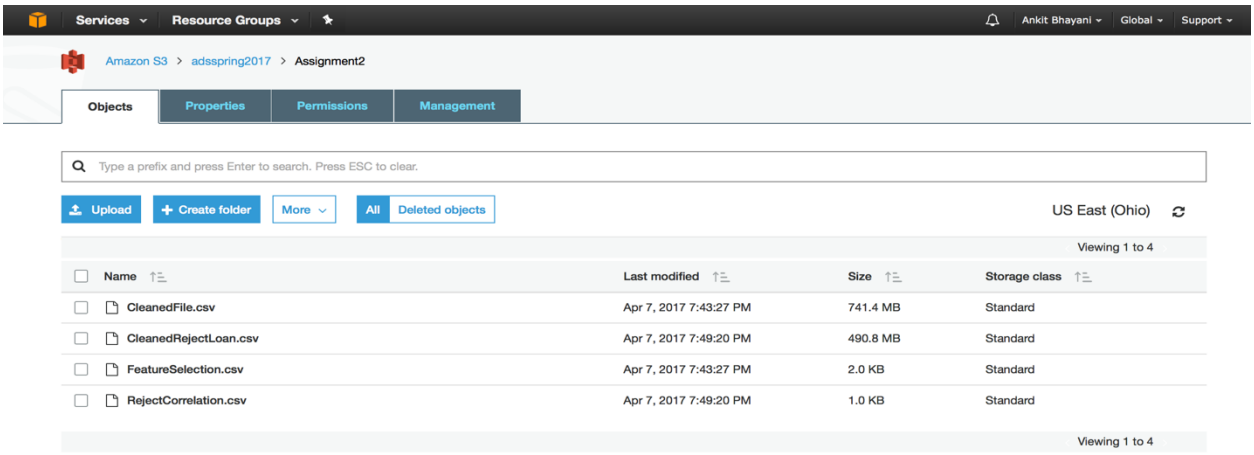
-e USERNAME=**<LENDINGCLUB_USERNAME>**

-e PASSWD=**< LENDINGCLUB_PASSWD >**

ankitbhayani/assignment2part1

/src/PART1-driver.sh declined





DRIVER CODE HANDLING THROUGH SHELL FILE:

```
#!/bin/bash

DECLINED=$1

if [ -z $DECLINED ]; then
    echo "Working on LendingClub Data-1"
    python3 /src/PART1-downloader_luigi.py featureSelection --local-scheduler --loginemail $USERNAME --loginpassword $PASSWD
    mv /src/CleanedFile.csv /src/Output/
    mv /src/FeatureSelection.csv /src/Output/
else
    echo "Working on Declined Data-2"
    python3 PART1-declined_luigi.py FeatureSelection --local-scheduler --loginemail $USERNAME --loginpassword $PASSWD
    mv /src/CleanedRejectLoan.csv /src/Output/
    mv /src/RejectCorrelation.csv /src/Output/
fi

find .

if [ $? -eq 0 ]
then
  echo "Successfully created the files now going to upload them in S3"
  sh /src/awsS3Upload.sh
else
  echo "Could not create file" >&2
fi
```

AWS S3 UPLOAD SHELL FILE:

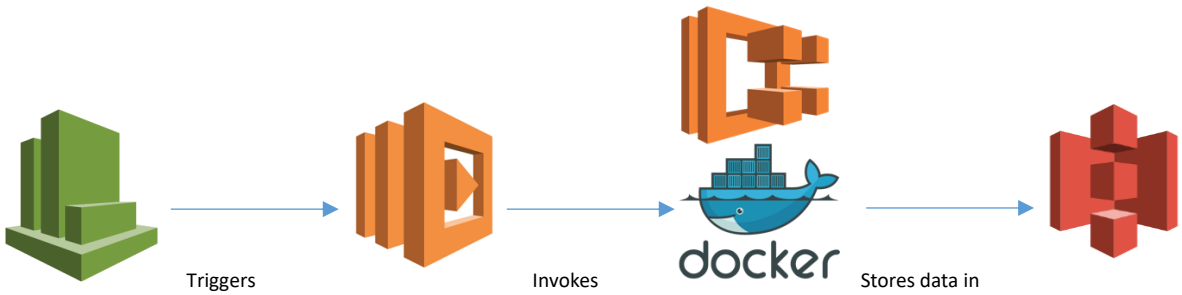
```
#!/bin/bash

set -e

: ${ACCESS_KEY:?"ACCESS_KEY env variable is required"}
: ${SECRET_KEY:?"SECRET_KEY env variable is required"}
: ${S3_PATH:?"S3_PATH env variable is required"}
: ${REGION:?"REGION env variable is required"}
export DATA_PATH=${DATA_PATH:-/data/}
#CRON_SCHEDULE=${CRON_SCHEDULE:-0 1 * * *}

echo "access_key=$ACCESS_KEY"
#>> /root/.s3cfg
echo "secret_key=$SECRET_KEY"
#>> /root/.s3cfg
echo "S3_PATH=$S3_PATH"
#>> /root/.s3cfg
echo "REGION=$REGION"

aws configure set aws_access_key_id $ACCESS_KEY
aws configure set aws_secret_access_key $SECRET_KEY
aws configure set default.region $REGION

echo "AWS S3 Job started: $(date)"

aws s3 sync /src/Output  $S3_PATH

echo "AWS S3 Job finished: $(date)"
```

RESEARCH ON SCHEDULING THE PIPELINE

Architecture:

When we need to execute a long-running or periodic jobs in AWS, it's been very conventional to write our scripts and configure them as **cron job** on EC2 instances. However, we risk the failure of EC2 instances and we may also have problem in its scalability.

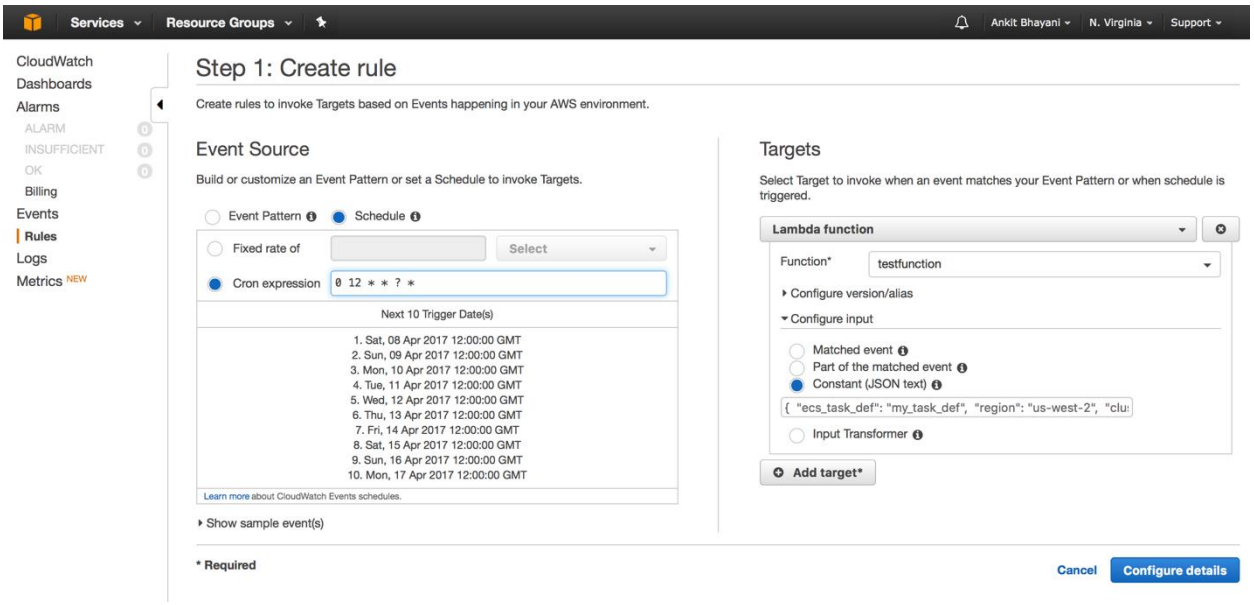
Amazon EC2 Container Service (ECS) is a highly scalable, high performance container management service that supports Docker containers and allows us to easily run applications on a managed cluster of Amazon EC2 instances. Amazon ECS eliminates the need for us to install, operate, and scale our own cluster management infrastructure. With simple API calls, we can launch and stop container-enabled applications, query the complete state of our cluster, and access many familiar features like security groups, Elastic Load Balancing, EBS volumes and IAM roles.



CloudWatch Lambda Function EC2 Container Service - ECS Amazon S3

In AWS **CloudWatch Events**, configure the schedule CRON expression as **0 12 * * ? *** and configure the constant JSON input payload like:

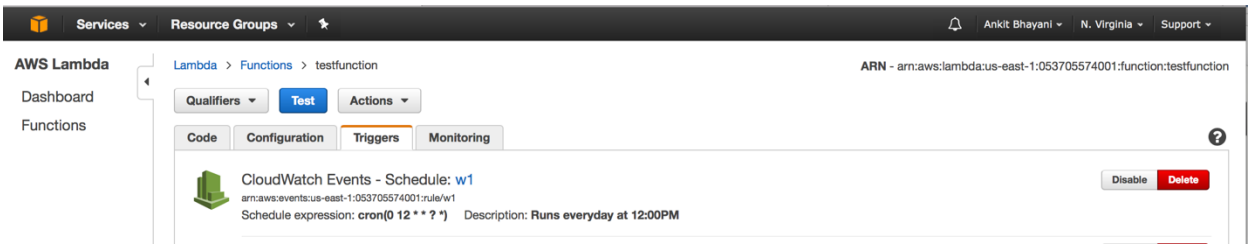
```
{
 "ecs_task_def": "my_task_def",
 "region": "us-west-2",
 "cluster": "default",
 "count": 1
 }
```



It will trigger an event at every day at 12:00PM. When invoking Lambda function, the input payload above will construct itself as the ***event*** object for Lambda, and Lambda will hence perform the one-shot task running on ECS as we expect. Following are some more examples of scheduling through cloudwatch events:

Minutes	Hours	Day of month	Month	Day of week	Year	Meaning
0	10	*	*	?	*	Run at 10:00 am (UTC) every day
15	12	*	*	?	*	Run at 12:15 pm (UTC) every day
0	18	?	*	MON-FRI	*	Run at 6:00 pm (UTC) every Monday through Friday
0	8	1	*	?	*	Run at 8:00 am (UTC) every 1st day of the month
0/15	*	*	*	?	*	Run every 15 minutes
0/10	*	?	*	MON-FRI	*	Run every 10 minutes Monday through Friday
0/5	8-17	?	*	MON-FRI	*	Run every 5 minutes Monday through Friday between 8:00 am and 5:55 pm (UTC)

AWS Lambda let us run the code without provisioning or managing servers (*server-less* fashion). We pay only for the compute time we consume - there is no charge when our code is not running. With Lambda, we can run code for virtually any type of application or backend service - all with zero administration. We just need to upload our code and Lambda takes care of everything required to run and scale our code with high availability. We can also set up our code to automatically trigger from other AWS services or call it directly from any web or mobile app.



Code(ecs-task-runner):

```
var AWS = require('aws-sdk')
var ecs = new AWS.ECS();
exports.handler = (events, context) => {
    // CLI example:
    // aws  --region ap-northeast-1 ecs run-task --task-definition my_task_def
    /* input event payload sample:
    {
       "ecs_task_def": "my_task_def",
       "region": "us-west-2",
       "cluster": "default",
       "count": 1
    }
    */

  console.log(events)
       var ecs_task_def = events.ecs_task_def || 'undefined'
       var exec_region  = events.region || 'ap-northeast-1'
       var cluster      = events.cluster || 'default'
       var count        = events.count || 1
  console.log(ecs_task_def, exec_region)
```

```
    var params = {
        taskDefinition: ecs_task_def,
        cluster: cluster,
        count: count
    }
    ecs.runTask(params, function(err, data) {
        if (err) console.log(err, err.stack); // an error occurred
        else     console.log(data);           // successful response
        context.done(err, data)
    })

}
```

An interesting approach is to wrap our business logic of periodic long-running jobs into Docker images and compile them into **AWS ECS Task Definitions** and use the **ECS CLI** or **SDK** to *run-task* with them. The ECS scheduler will try to allocate resources for the task on proper ECS instance and execute the Docker command we wrapped in.

ECS CLI

Since we already wrap our codes of business logic into the docker image and created our ECS task definition. It's very easy to run the task for one-shot just like this:

```
aws --region ap-northeast-1 ecs run-task --task-definition my_task_definition
```

ECS Console - *Docker Repository on ECS*

Register your docker image with ECS with following commands:

1) Retrieve the docker login command that you can use to authenticate your Docker client to your registry:
   ```
   aws ecr get-login --region us-east-1
   ```
2) Run the docker login command that was returned in the previous step.

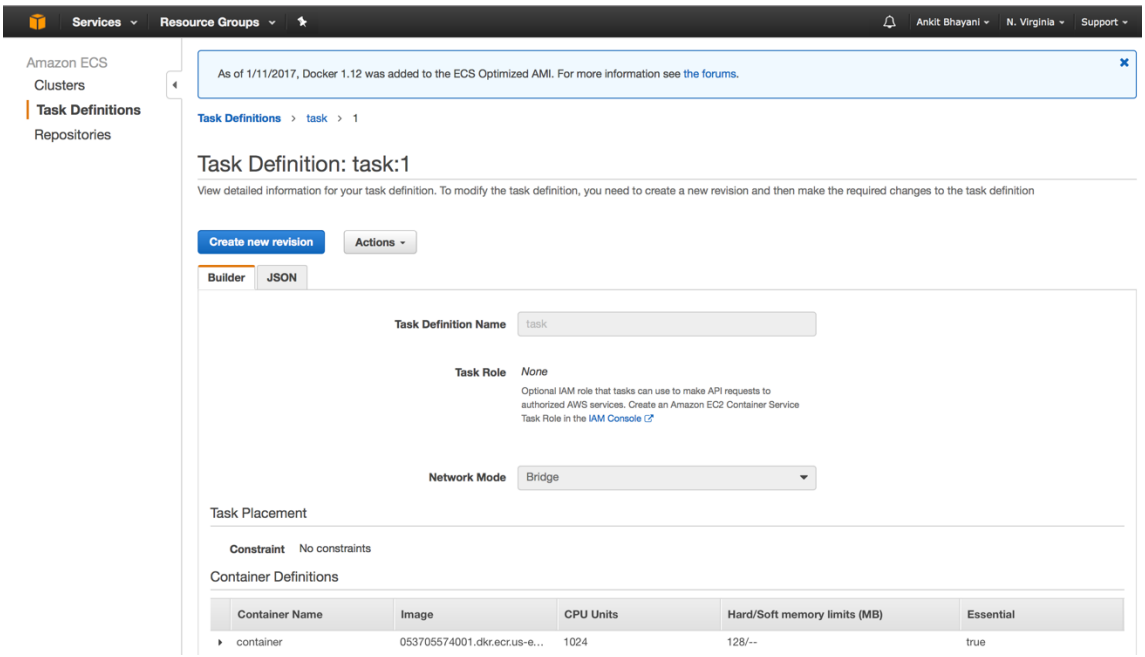
3) Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:
   ```
   docker build -t dockerrepository .
   ```

4) After the build completes, tag your image so you can push the image to this repository:
   ```
   docker tag dockerrepository:latest 053705574001.dkr.ecr.us-east-
   1.amazonaws.com/dockerrepository:latest
   ```

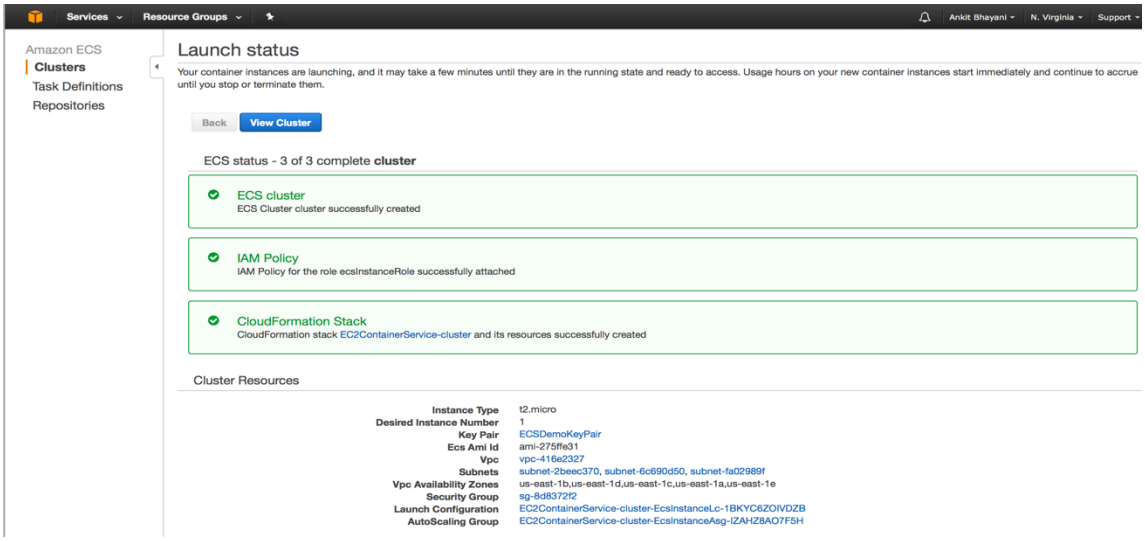
5) Run the following command to push this image to your newly created AWS repository:
   ```
   docker push 053705574001.dkr.ecr.us-east-1.amazonaws.com/dockerrepository:latest
   ```

Repository Sceenshot which contains latest image pushed from the terminal.

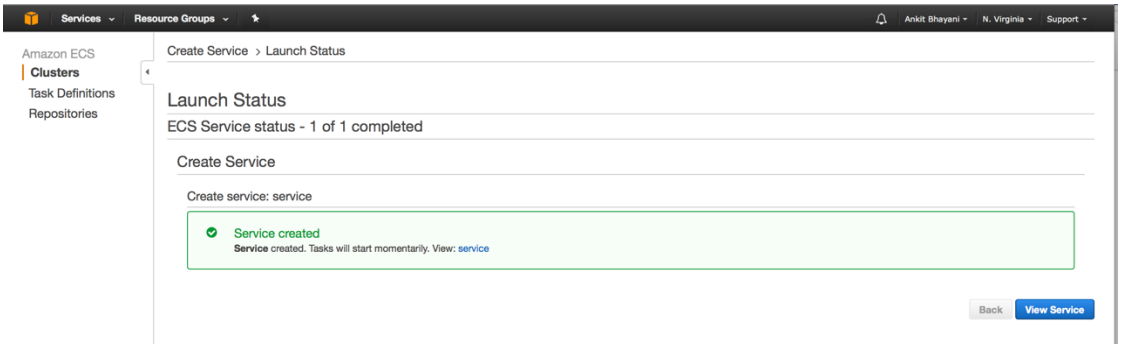
Creating the task which will be executed on ec2 container.



Creating a cluster which will provide the infrastructure to run above task and service

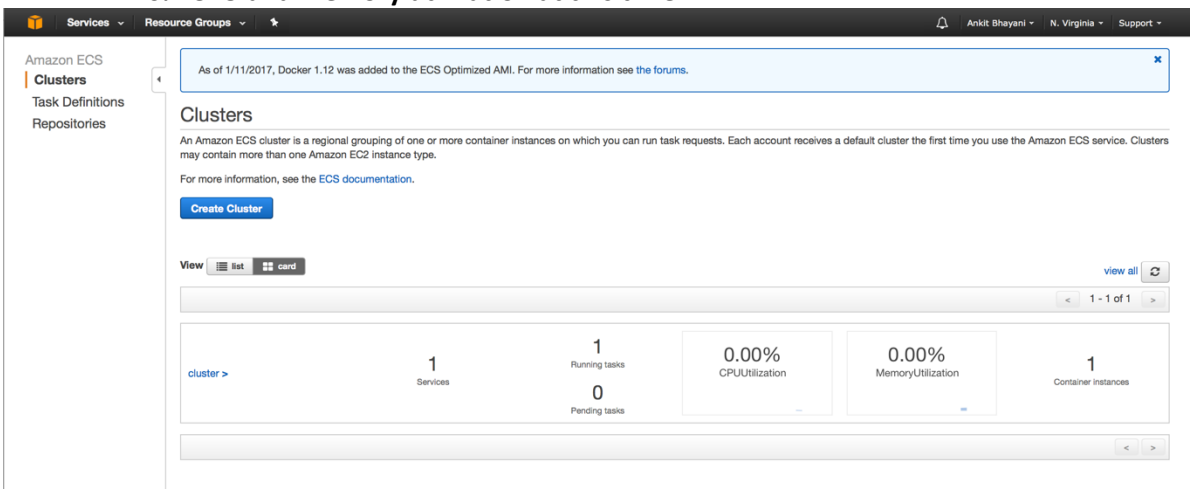


Create service

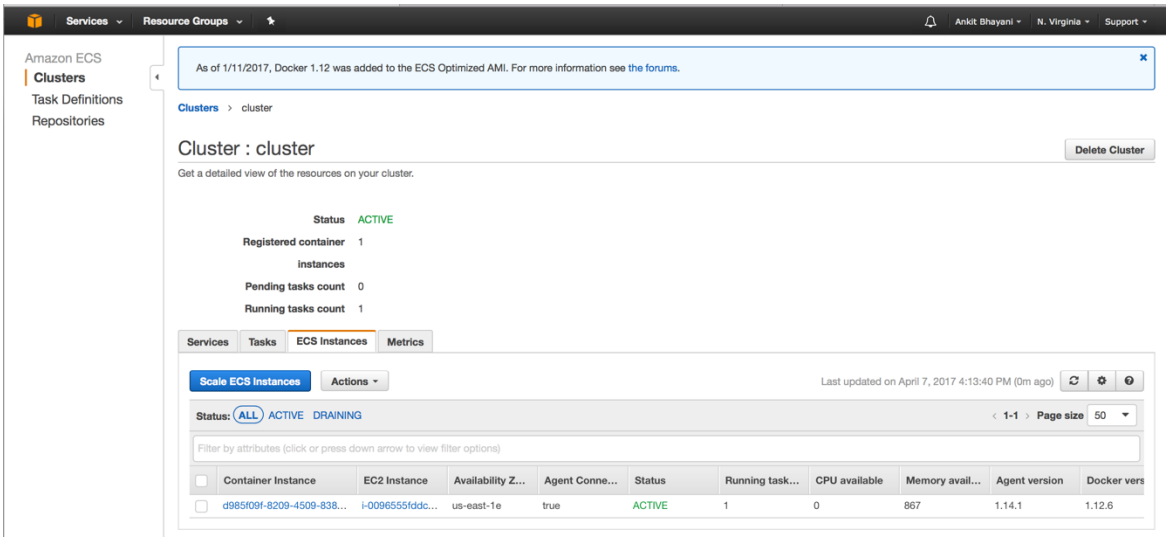


Cluster started with

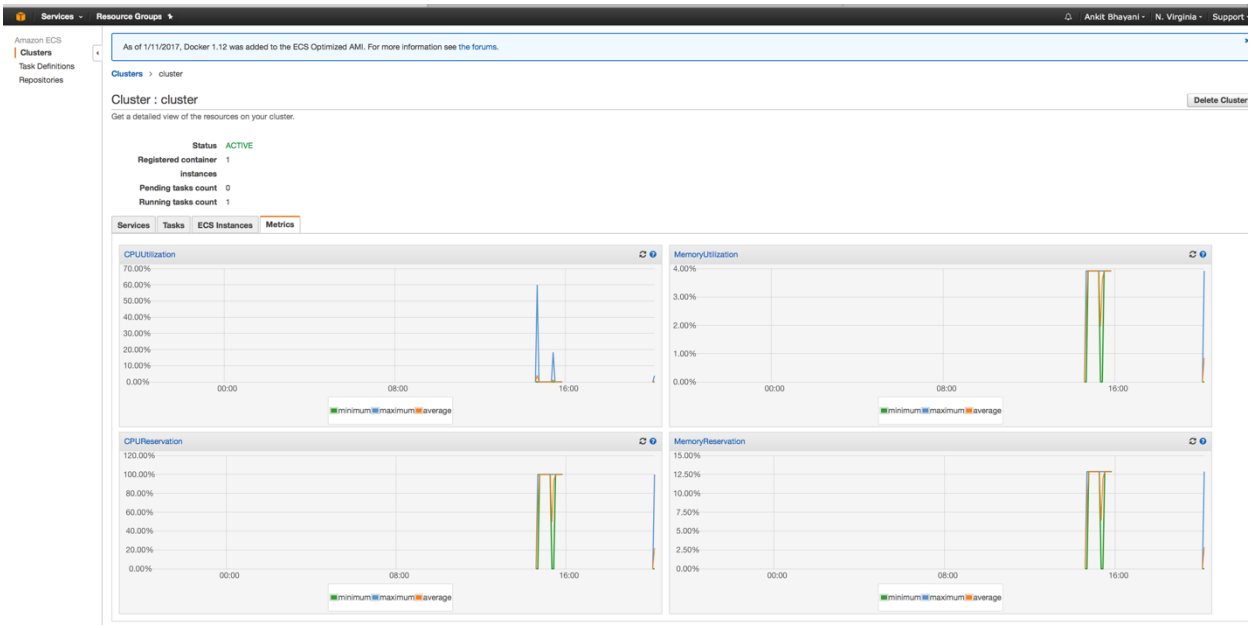
- **1 service, 1 Running task and 1 container instance**
- **0% CPU and memory utilization at this time**

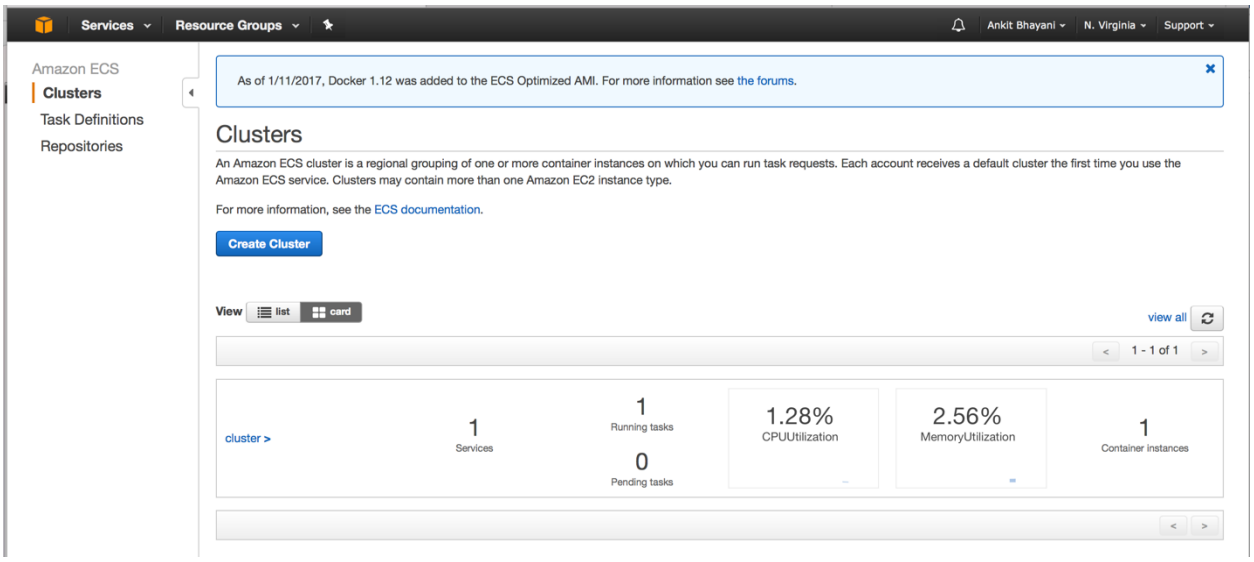


ECS instance invoked



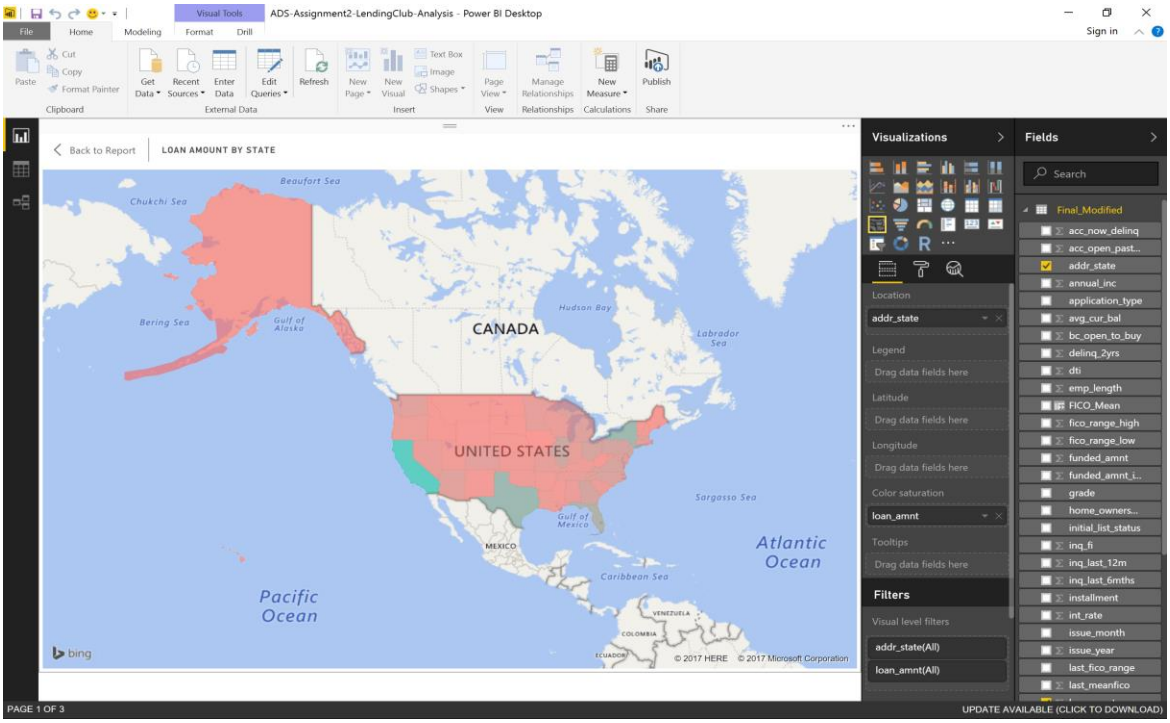
Cluster Metrics for CPU and memory utilization after a while

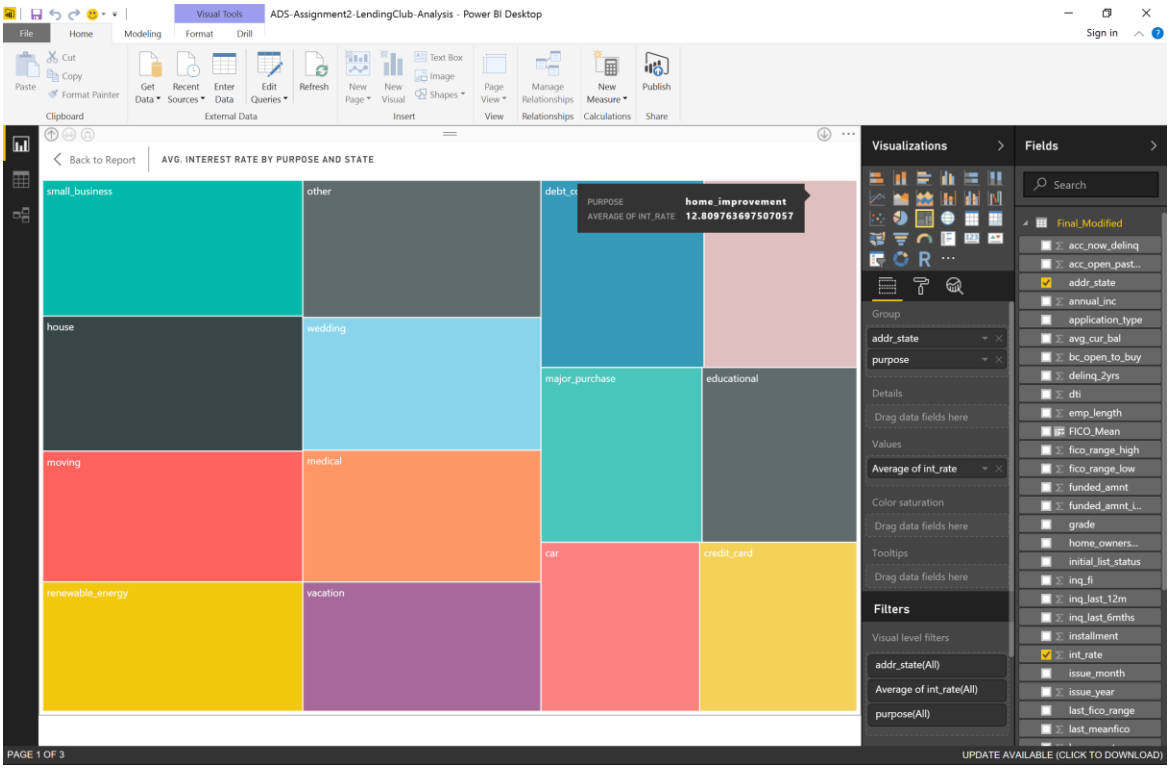


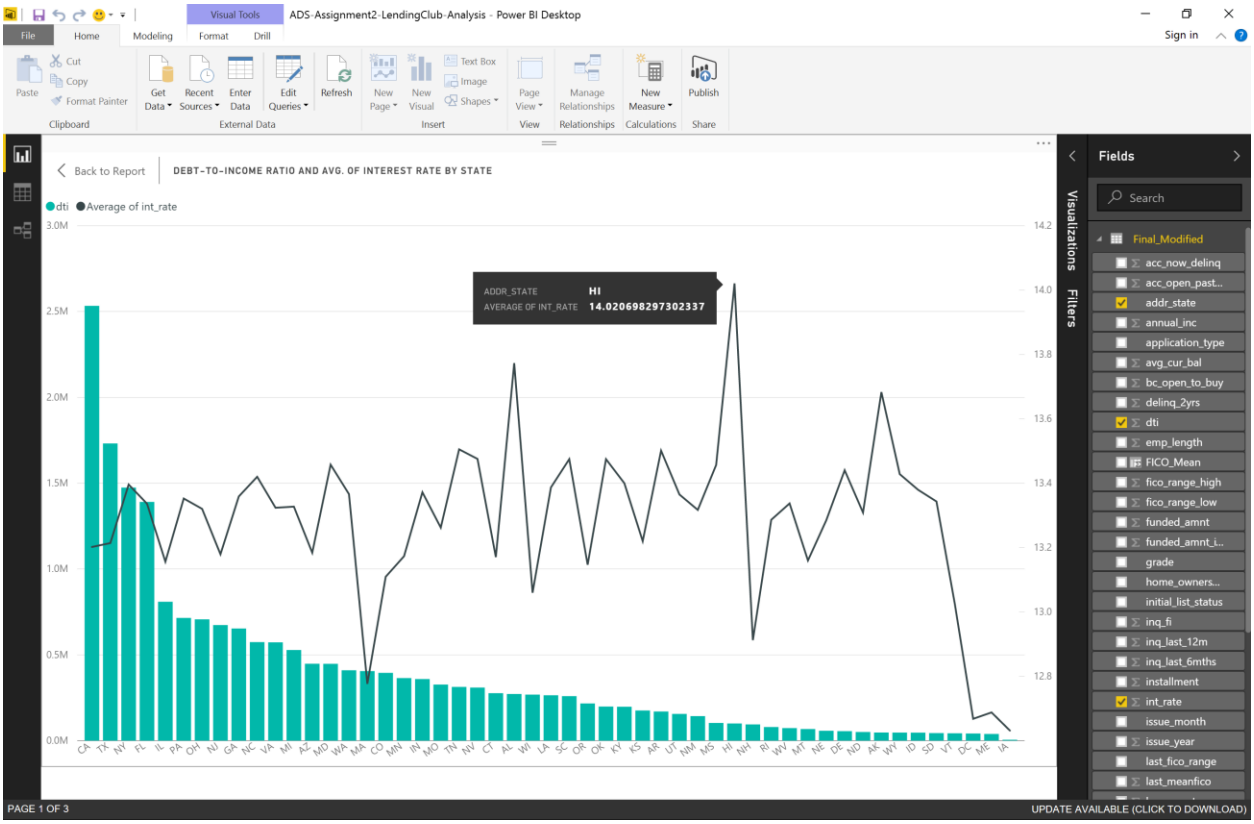


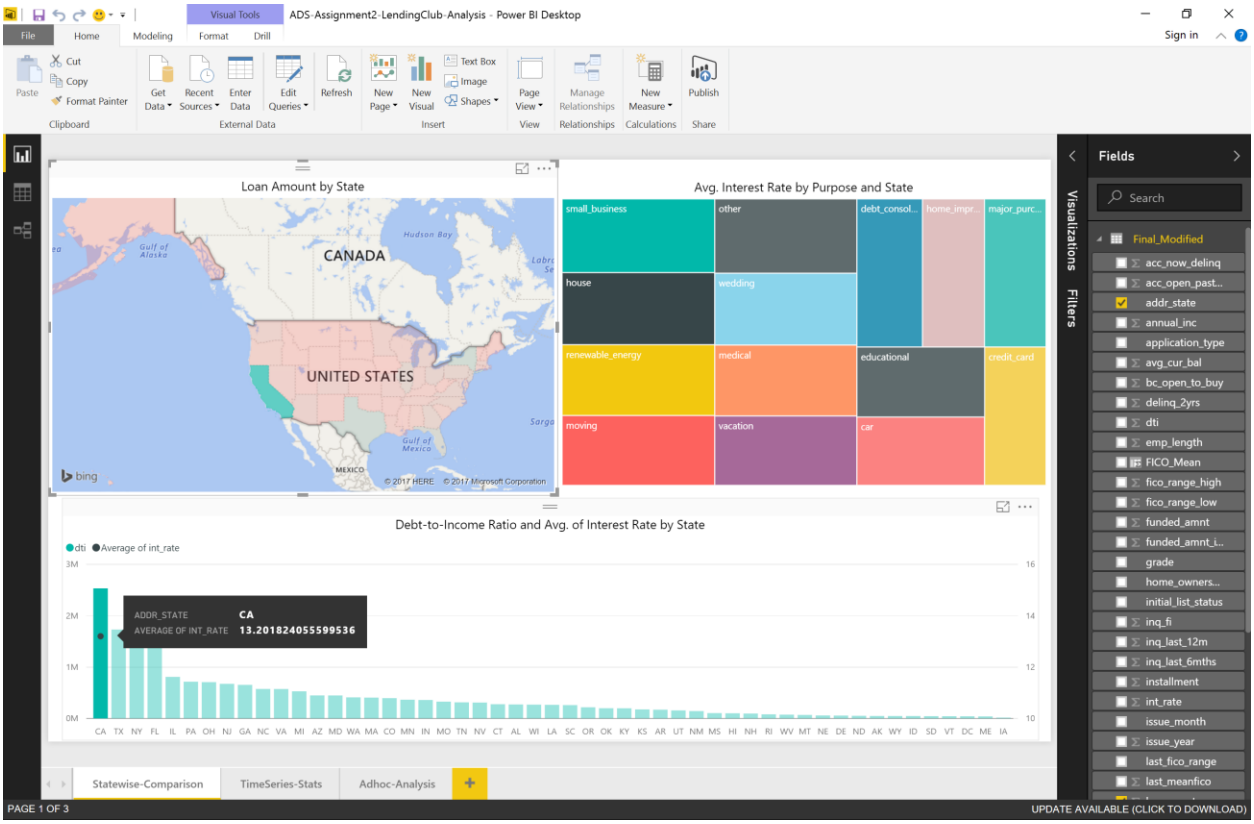
ANALYSIS THROUGH POWER BI

STATEWISE Comparison

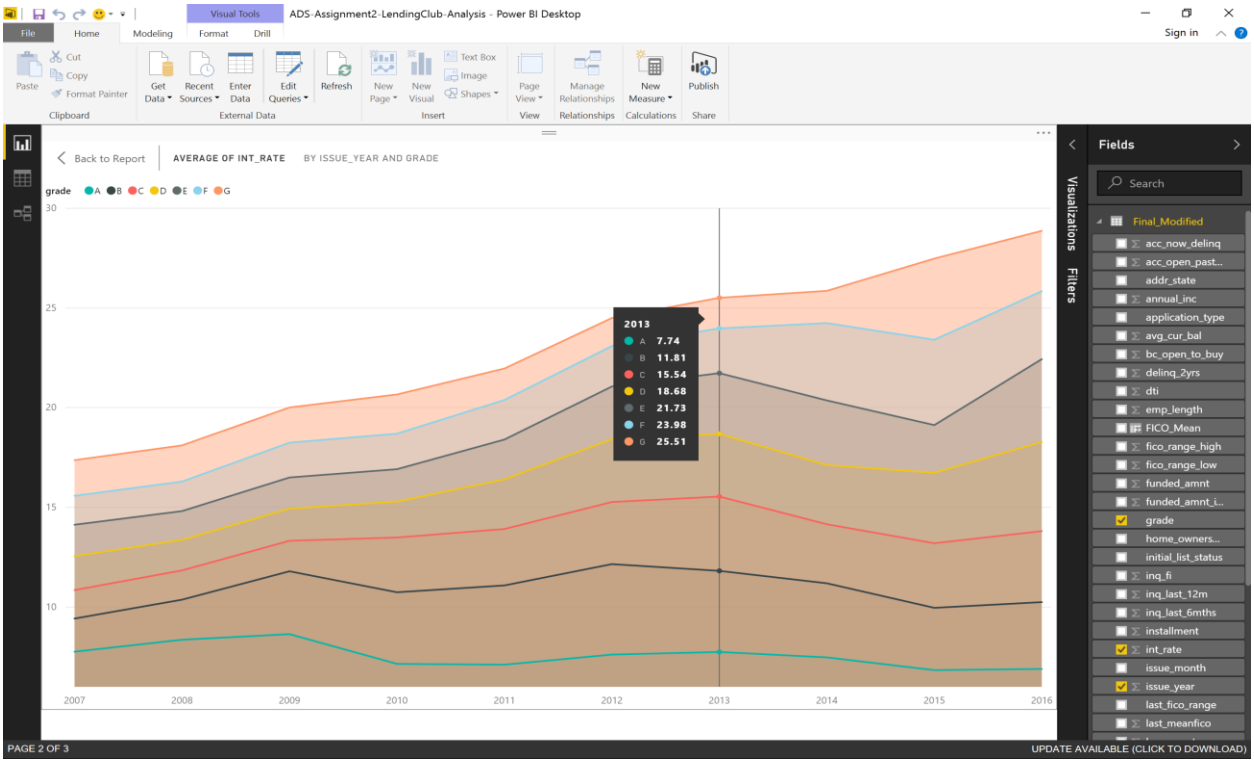


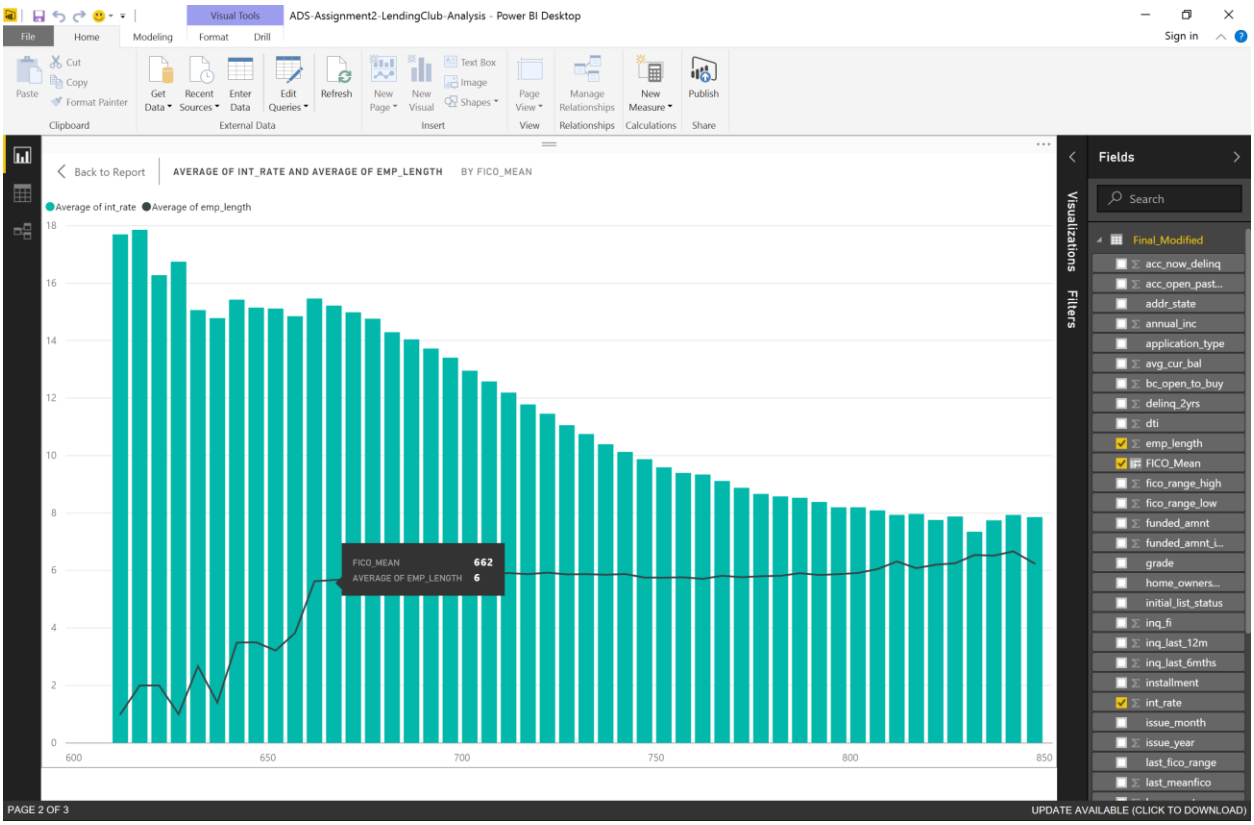


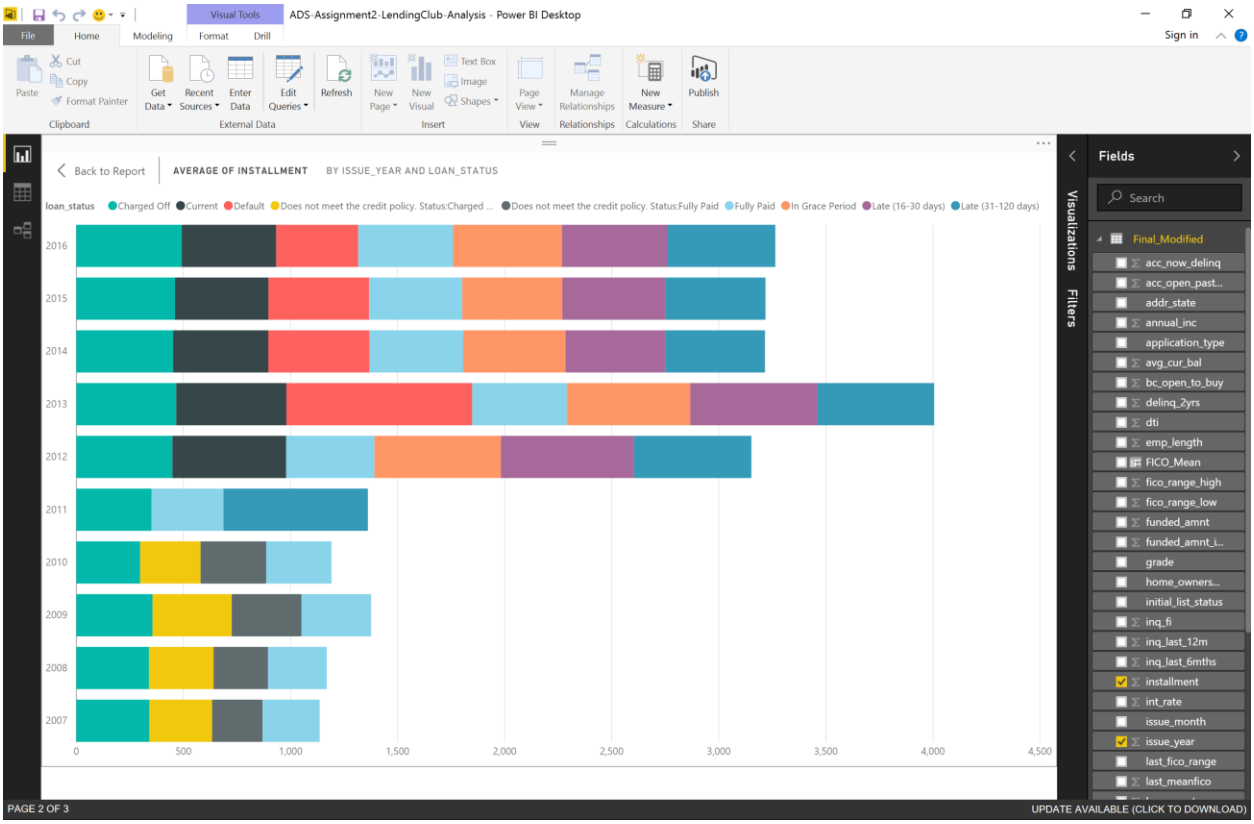

ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Back to Report
DEBT-TO-INCOME RATIO AND AVG. OF INTEREST RATE BY STATE
dti
Average of int_rate
ADDR_STATE HI
AVERAGE OF INT_RATE 14.020698297302337
Fields
Search
Final_Modified
PAGE 1 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)

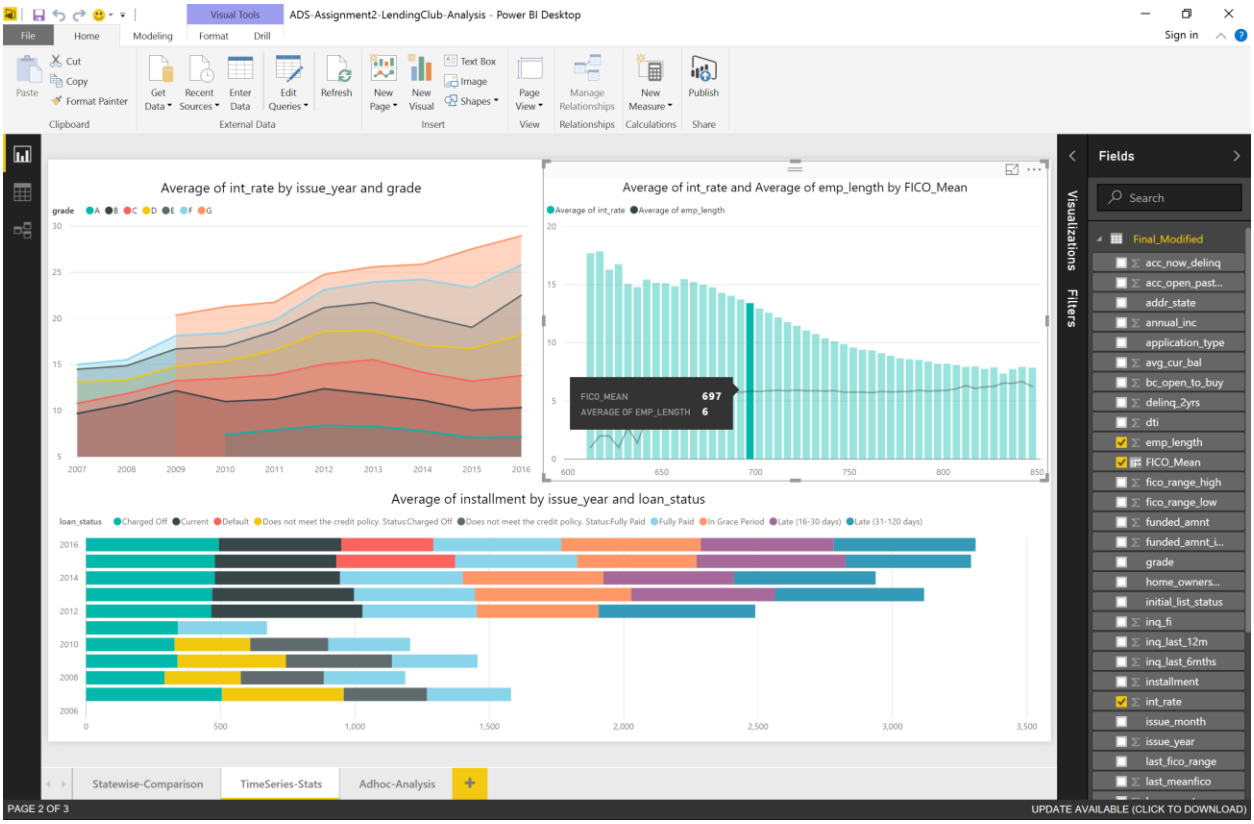

ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Loan Amount by State
CANADA
UNITED STATES
MEXICO
Avg. Interest Rate by Purpose and State
small_business
other
house
wedding
renewable_energy
medical
educational
moving
vacation
car
Debt-to-Income Ratio and Avg. of Interest Rate by State
ADDR_STATE CA
AVERAGE OF INT_RATE 13.201824055599536
Statewise-Comparison
TimeSeries-Stats
Adhoc-Analysis
Fields
PAGE 1 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)

TIME-Series Stats

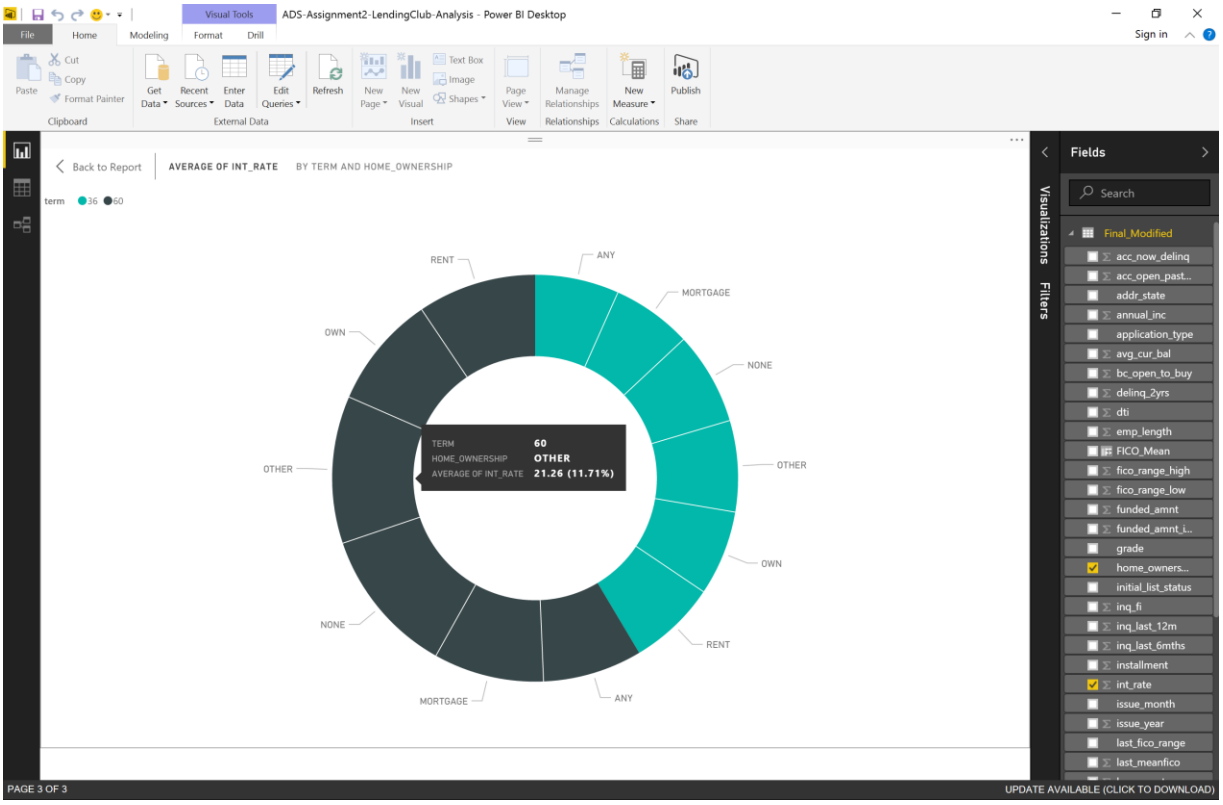


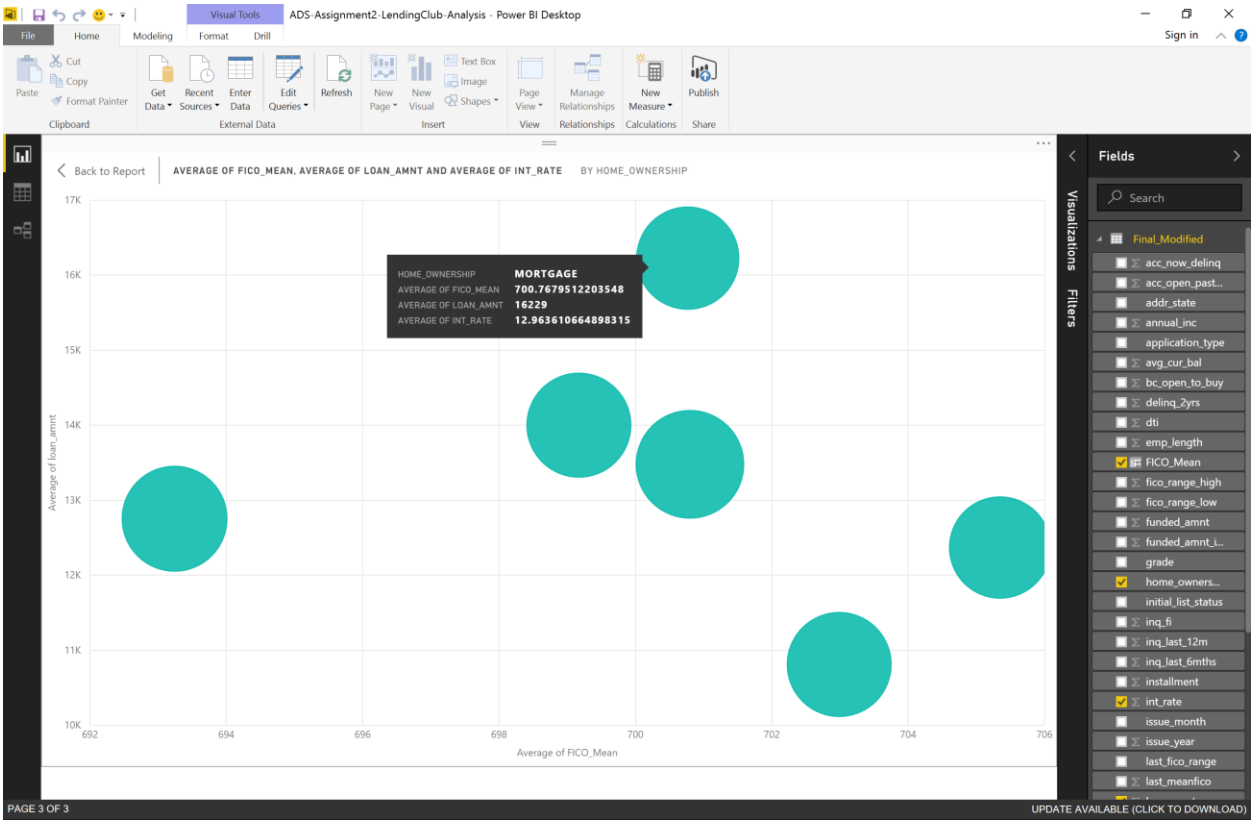


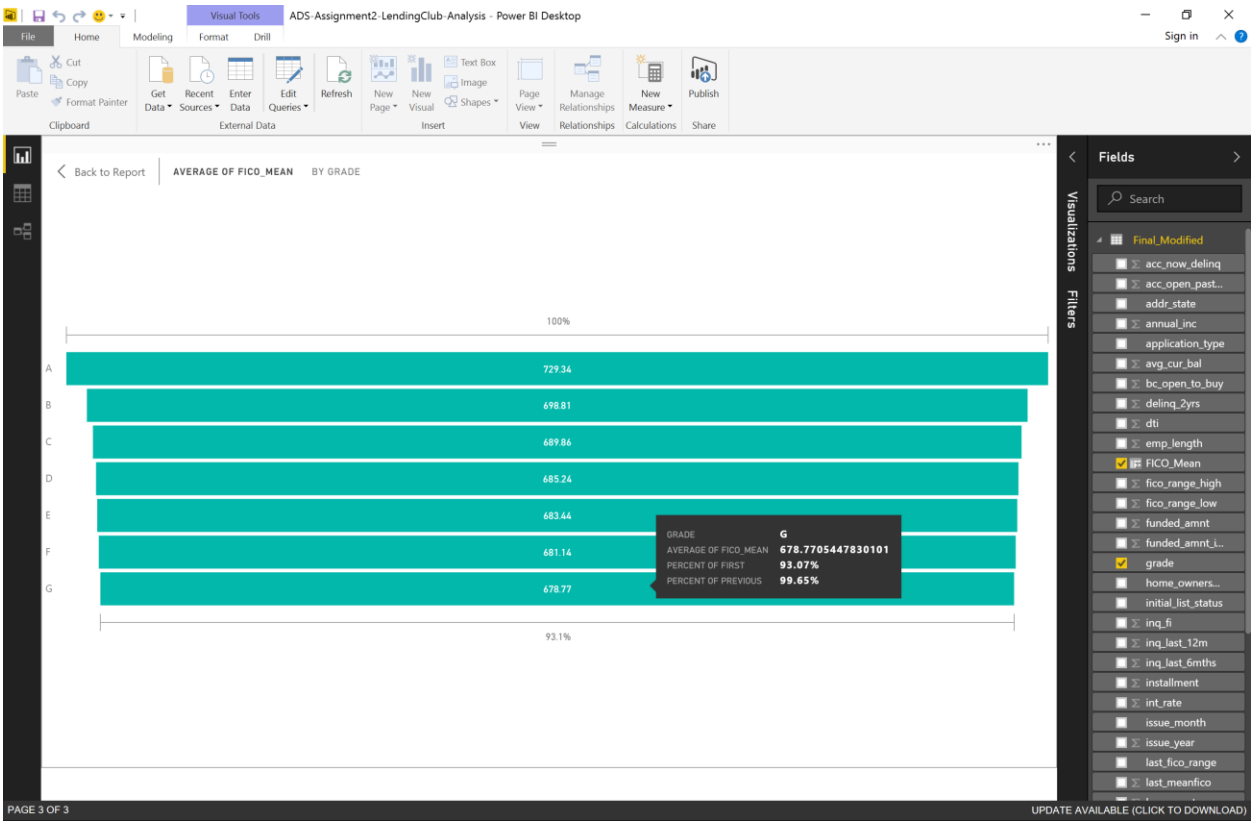

ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Back to Report
AVERAGE OF INSTALLMENT BY ISSUE_YEAR AND LOAN_STATUS
loan_status Charged Off Current Default Does not meet the credit policy. Status:Charged ... Does not meet the credit policy. Status:Fully Paid Fully Paid In Grace Period Late (16-30 days) Late (31-120 days)
Fields
Search
Final_Modified
PAGE 2 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)

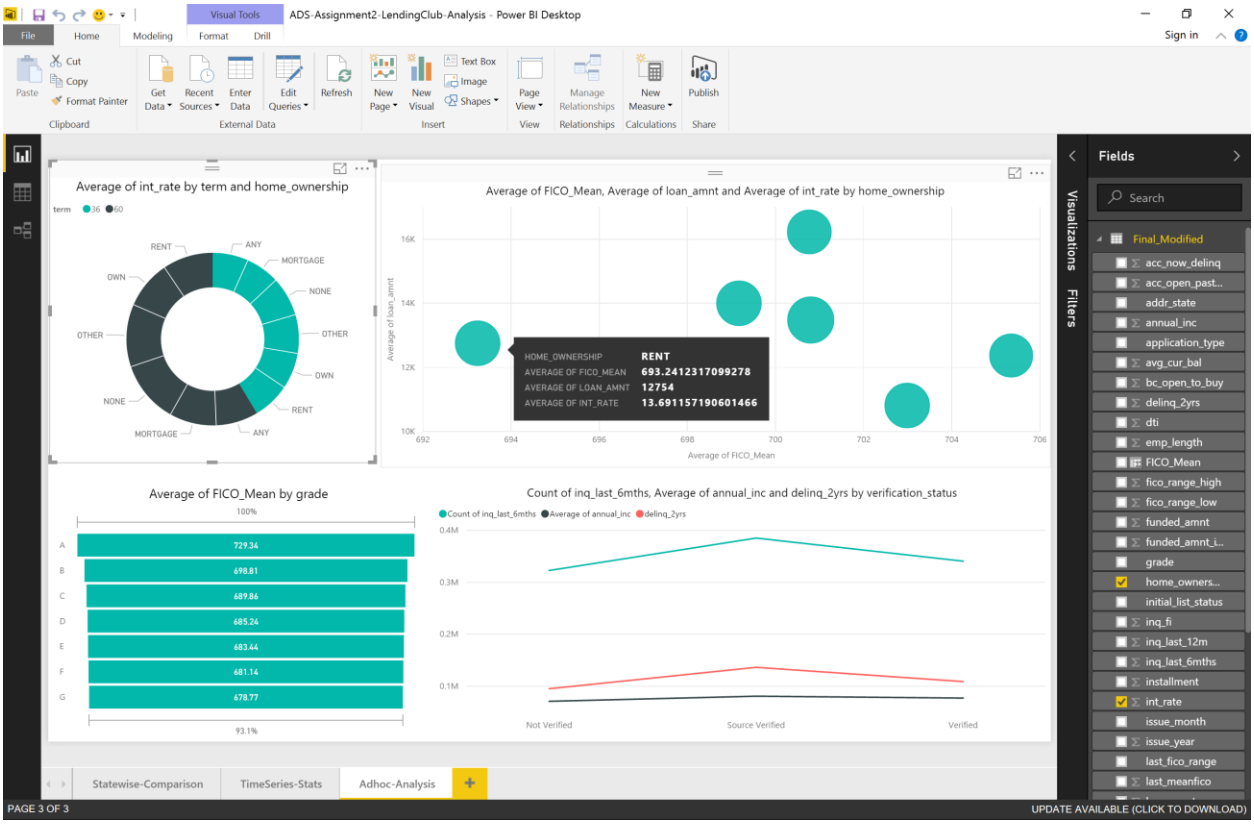

ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Average of int_rate by issue_year and grade
Average of int_rate and Average of emp_length by FICO_Mean
FICO_MEAN 697
AVERAGE OF EMP_LENGTH 6
Average of installment by issue_year and loan_status
Statewise-Comparison
TimeSeries-Stats
Adhoc-Analysis
Fields
Search
PAGE 2 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)

ADHOC- Analysis






ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Back to Report
AVERAGE OF FICO_MEAN BY GRADE
100%
A 729.34
B 698.81
C 689.86
D 685.24
E 683.44
F 681.14
G 678.77
93.1%
GRADE G
AVERAGE OF FICO_MEAN 678.7705447830101
PERCENT OF FIRST 93.07%
PERCENT OF PREVIOUS 99.65%
Fields
Final_Modified
PAGE 3 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)


ADS-Assignment2-LendingClub-Analysis - Power BI Desktop
Average of int_rate by term and home_ownership
Average of FICO_Mean, Average of loan_amnt and Average of int_rate by home_ownership
HOME_OWNERSHIP RENT
AVERAGE OF FICO_MEAN 693.2412317099278
AVERAGE OF LOAN_AMNT 12754
AVERAGE OF INT_RATE 13.691157190601466
Average of FICO_Mean by grade
Count of inq_last_6mths, Average of annual_inc and delinq_2yrs by verification_status
Statewise-Comparison
TimeSeries-Stats
Adhoc-Analysis
Fields
PAGE 3 OF 3
UPDATE AVAILABLE (CLICK TO DOWNLOAD)

Interest Rate Based on Input: