1. **What is the problem you want to solve?**

Banks make a significant portion of their money in the form of interest from loans.  In addition to many of the other risks that banks face from inflation to interest rate, the most significant risk on a loan is default risk.  The risk that the payer of the loan is unable to pay the full amount of the loan. Thus the ability to foresee who these customers are is valuable in that we can charge them more in order to adequately account for the risk, or deny them completely.

1. **Who is your client and why do they care about this problem? In other words, what will your client DO or DECIDE based on your analysis that they wouldn’t have otherwise?**

The client would be any variety of firms that hand out loans to potential customers from large banks to small businesses like car dealerships and medical companies looking to keep loans ‘in-house’.  Clients will be able to use this information to decide what interest rates should be for customers given a set of input information. They may also decide if they want to take certain customers at all.

1. **Describe and explain how I cleaned the dataset**

LendingClub is a website that allows individuals and institutions to either apply for loans or sell loans.  The data acquired as a result of these loans from 2007 to present is publicly available here: <https://www.lendingclub.com/info/download-data.action>.  Since some of the data is split up into multiple sheets, we will have to recombine these files to apply algorithms to all these data points.

**Data Wrangling:**

Drop all the loans that were not last the due date.  Convert appropriate columns to datetime objects.

#sort out the completed loans and convert to datetimes

completed\_2015 = l\_2015.loc[l\_2015['term']==' 36 months']

completed\_2015['issue\_d'] = pd.to\_datetime(completed\_2015['issue\_d'])

completed\_2015['earliest\_cr\_line'] = pd.to\_datetime(completed\_2015['earliest\_cr\_line'])

completed\_2015 = completed\_2015.loc[completed\_2015['issue\_d'] <= pd.to\_datetime('5/1/2015')]

completed\_2014 = l\_2014.loc[l\_2014['term']==' 36 months']

completed\_2014['issue\_d'] = pd.to\_datetime(completed\_2014['issue\_d'])

completed\_2014['earliest\_cr\_line'] = pd.to\_datetime(completed\_2014['earliest\_cr\_line'])

short\_2012\_2013 = l\_2012\_2013.loc[l\_2012\_2013['term']==' 36 months']

long\_2012\_2013 = l\_2012\_2013.loc[l\_2012\_2013['term']==' 60 months']

short\_2012\_2013['issue\_d'] = pd.to\_datetime(short\_2012\_2013['issue\_d'])

long\_2012\_2013['issue\_d'] = pd.to\_datetime(long\_2012\_2013['issue\_d'])

long\_2012\_2013 = long\_2012\_2013.loc[long\_2012\_2013['issue\_d'] <= pd.to\_datetime('5/1/2013')]

completed\_2012\_2013 = pd.concat([short\_2012\_2013, long\_2012\_2013])

completed\_2012\_2013['earliest\_cr\_line'] = pd.to\_datetime(completed\_2012\_2013['earliest\_cr\_line'])

l\_2007\_2011['issue\_d'] = pd.to\_datetime(l\_2007\_2011['issue\_d'])

l\_2007\_2011['earliest\_cr\_line'] = pd.to\_datetime(l\_2007\_2011['earliest\_cr\_line'])

before\_2015 = [l\_2007\_2011, completed\_2012\_2013, completed\_2014, completed\_2015]

Selected columns that I plan to use to predict defaults and combine datasets together

rev\_cols = ['loan\_amnt', 'funded\_amnt', 'term', 'grade', 'sub\_grade', 'installment', 'emp\_length', 'home\_ownership', 'int\_rate',  \

          'annual\_inc', 'verification\_status', 'issue\_d', 'loan\_status', 'purpose', 'addr\_state', 'dti', 'delinq\_2yrs', \

          'earliest\_cr\_line', 'inq\_last\_6mths', 'open\_acc', 'pub\_rec', 'total\_acc', \

           'policy\_code', 'application\_type', 'revol\_bal', 'revol\_util']

#reduce columns and combine together.

before\_2015\_reduced = [loan[rev\_cols] for loan in before\_2015]

completed = pd.concat(before\_2015\_reduced, axis=0)

Remove the percentage sign and store percentages as decimals instead.

#remove percentage sign and convert to decimal floats

def stripper(x):

   y = np.nan

   if x != np.nan:

       try:

           y = float(float(x.strip('%'))/100)

       except:

           y = y

   return y

completed['int\_rate'] = completed.int\_rate.apply(stripper)

completed['revol\_util'] = completed.revol\_util.apply(stripper)

Grouped interest rates and default rates by grade

#interest and default rates by subgroup

group\_interest = completed.groupby(['grade','term'])['int\_rate'].mean()

temp1 = completed.groupby(['grade','term'])['default'].count()

temp2 = completed.groupby(['grade','term'])['default'].sum()

temp3 = temp2/temp1

group\_interest = pd.concat([group\_interest, temp3,temp1], axis=1)

group\_interest.columns = ['int\_rate','default\_rate','count']

group\_interest

Group other columns by subgroups default rates and counts

def find\_default(column):

   #interest and default rates by subgroup

   temp1 = completed.groupby([column])['default'].count()

   temp2 = completed.groupby([column])['default'].sum()

   temp3 = temp2/temp1

   group = pd.concat([temp3,temp1], axis=1)

   group.columns = ['default\_rate','count']

   return group

find\_default('emp\_length')

find\_default('home\_ownership')

find\_default('inq\_last\_6mths')

find\_default('verification\_status')

find\_default('purpose')

find\_default('delinq\_2yrs')

Bucket Credit Inquiries into several different subgroups to plot.

inq = find\_default('inq\_last\_6mths')

inq\_group = pd.DataFrame(index=['0','1','2','3','4','5','6','7+'], columns=['default\_rate'])

inq\_group['default\_rate']['0']=inq['default\_rate'][0]

inq\_group['default\_rate']['1']=inq['default\_rate'][1]

inq\_group['default\_rate']['2']=inq['default\_rate'][2]

inq\_group['default\_rate']['3']=inq['default\_rate'][3]

inq\_group['default\_rate']['4']=inq['default\_rate'][4]

inq\_group['default\_rate']['5']=inq['default\_rate'][5]

inq\_group['default\_rate']['6']=inq['default\_rate'][6]

inq\_group['default\_rate']['7+']=inq['default\_rate'][7:].mean()

Bucket delinquent payments in the last 2 years into several groups

delinq = find\_default('delinq\_2yrs')

delinq\_group = pd.DataFrame(index=['0','1','2','3','4','5','6','7+'], columns=['default\_rate'])

delinq\_group['default\_rate']['0']=delinq['default\_rate'][0]

delinq\_group['default\_rate']['1']=delinq['default\_rate'][1]

delinq\_group['default\_rate']['2']=delinq['default\_rate'][2]

delinq\_group['default\_rate']['3']=delinq['default\_rate'][3]

delinq\_group['default\_rate']['4']=delinq['default\_rate'][4]

delinq\_group['default\_rate']['5']=delinq['default\_rate'][5]

delinq\_group['default\_rate']['6']=delinq['default\_rate'][6]

delinq\_group['default\_rate']['7+']=delinq['default\_rate'][7:].mean()

Remove outliers to clarify plotting

#remove outliers to see histograms in better detail

def outlierless(string, num):

   df = completed.loc[completed[string] <= num][[string, 'default']]

   return df

less\_annual = outlierless('annual\_inc', 200000)

less\_open = outlierless('open\_acc', 60)

less\_pub = outlierless('pub\_rec', 6)

less\_revolb = outlierless('revol\_bal', 100000)

less\_revolu = outlierless('revol\_util', 2)

1. **What are some other datasets you could use?**

Lending Club is not the only source out there for public loan data.  The Federal Housing Finance Agency lists data on mortgages acquired by Fannie\_Mae and Freddie\_Mac.  Fannie\_Mae also lists loans by itself. A search of Data.gov for loans found almost 500 results. There is no shortage of loan data on the web.

**Initial Findings:**

The dataset had many empty and sparse columns that were either added on or later on or were used for when collection processes began.  Many of the columns that used information collected before issuing the loan did tell the issuer much about the probability of a charge off.  The best predictors are Grade, Purpose, and Inquiries in the Last 6 Months.