Name: Morgan Gere
Course:IST 652- Scripting for Data Analytics
Instructor: Dr Debbie Landowski
Date: 3/24/2022

Loans

Introduction

During this project I worked by myself to learn as much as possible with Python. The project looked at loans, attempting to find insight into loan defaulting. Analysis was conducted to gain information into reasons why loans default and what are good predictors of defaulting. During this initial analysis I found that there was an increased amount of small business loans defaulting compared to other loan types. Using Tweets obtained about businesses I attempted to find a way to indicate if a company was at higher risk of bankruptcy or preforming bad business practices. This could help cut down on the number of defaulted small business loans issued.

Data Sources

The data can be obtained from Lending Club Loan Data Analysis | Kaggle (https://www.kaggle.com/urstrulyvikas/lending-club-loan-data-analysis). It consists of 751.25 kB being made up of 14 columns and 9578 rows. The columns are specific information about both past events and the loan the individual is receiving. The data was read into a pandas data frame.

The second data source was obtained using snscrape. A dictionary of each tweet, and tweet information, was placed into a list, this list was stored into mongodb so the same tweets can be used throughout the coding process. The list of dictionaries was pulled from the mongo data base and placed into a pandas data frame. The contents of the tweets were pulled and placed into a list since this was what was used throughout the process. There were 10,001 tweets obtained based off business names. This process was repeated three times. Dunkin Donuts, Starbucks and Toys-R-Us were the business chosen. Toys-R-Us was selected because it is a business that went bankrupt recently, meaning there should be tweets that indicate the company would default on any loans given to them.

Questions and topics

1. How does the average monthly amount due change in loans that were paid back vs loans that were not paid?
2. Does the purpose of the loan change the rate of being paid back or not?
3. A. Does annual income change the percentage of loans being paid back?

   B. *What annual income ranges produces the least amount of defaulted small business loans?
4. A. Does more debt-to-income ratio change the percentage of loans being paid back?

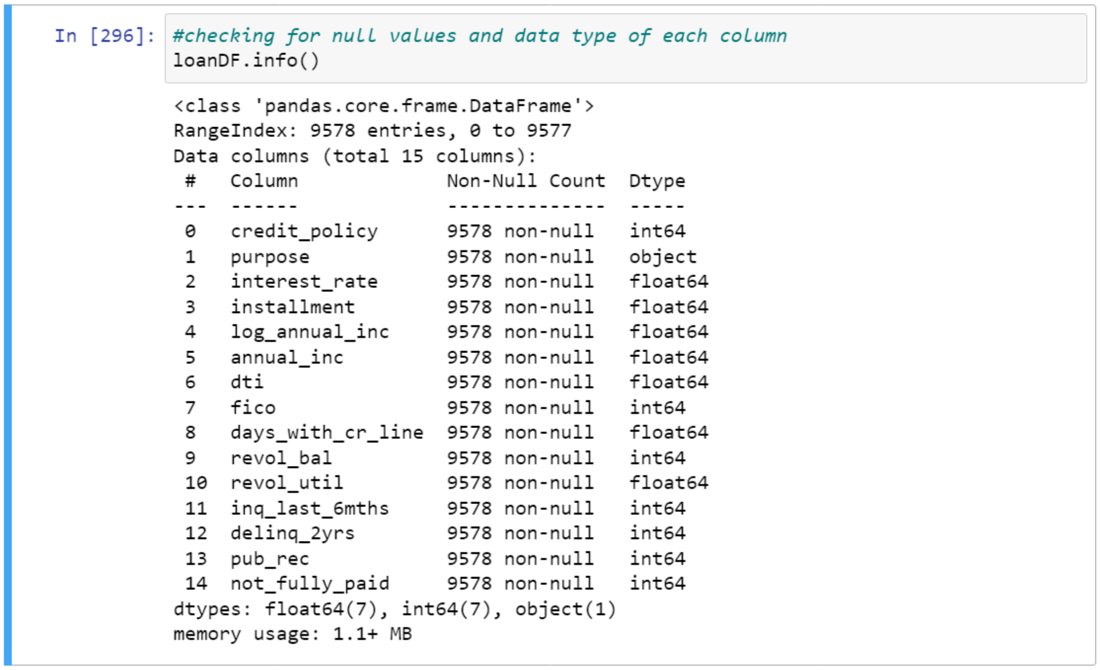
   B. * How does debt-to-income ratio change with small business loans?
5. A. How does fico credit score change the percentage of loans being paid vs not?

   B. *What fico credit rating range produces the least amount of defaulted small business loans?
6. *Can sentiment analysis be used to increase knowledge and predict if a small business loan will default?
7. *Can regular expressions be used to increase knowledge and predict if a small business loan will default?
8. *Can machine learning provide a way to predict if an applicant will default on a loan?
9. *Was anything found that significantly increase the chance that a small business loan does not default?

*- Represents questions that were not answered in Homework one or two.

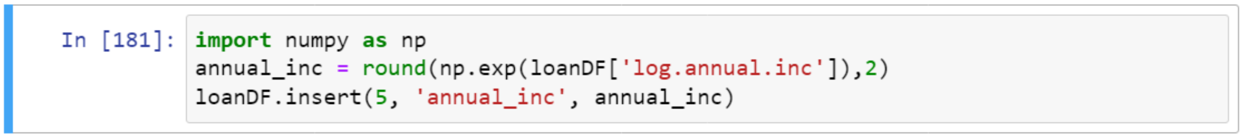
Reading and cleaning the Data (Kaggle loan Data set)

The Lending club loan data (which will be referred as loan data) was read in using Pandas. After first inspection there was found to be no NAN values and the columns appear to be in proper type.



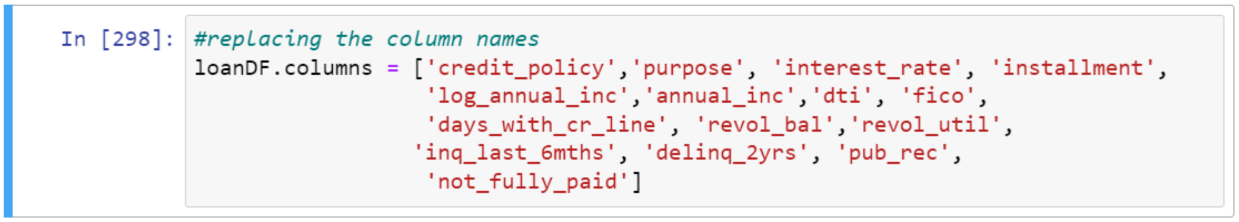
```
In [296]: #checking for null values and data type of each column
          loanDF.info()

          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 9578 entries, 0 to 9577
          Data columns (total 15 columns):
           #   Column             Non-Null Count  Dtype
          ---  ------             --------------  -----
           0   credit_policy      9578 non-null   int64
           1   purpose            9578 non-null   object
           2   interest_rate      9578 non-null   float64
           3   installment        9578 non-null   float64
           4   log_annual_inc     9578 non-null   float64
           5   annual_inc         9578 non-null   float64
           6   dti                9578 non-null   float64
           7   fico               9578 non-null   int64
           8   days_with_cr_line  9578 non-null   float64
           9   revol_bal          9578 non-null   int64
           10  revol_util         9578 non-null   float64
           11  inq_last_6mths     9578 non-null   int64
           12  delinq_2yrs        9578 non-null   int64
           13  pub_rec            9578 non-null   int64
           14  not_fully_paid     9578 non-null   int64
          dtypes: float64(7), int64(7), object(1)
          memory usage: 1.1+ MB
```

There was a column that was log.annual.inc which had a description of, "The natural log of the self-reported annual income of the borrower.". This column will be useful later for machine learning, as for now, it is hard for visual understanding, so it was translated back to its original base. This was added as a new column named annual_inc.



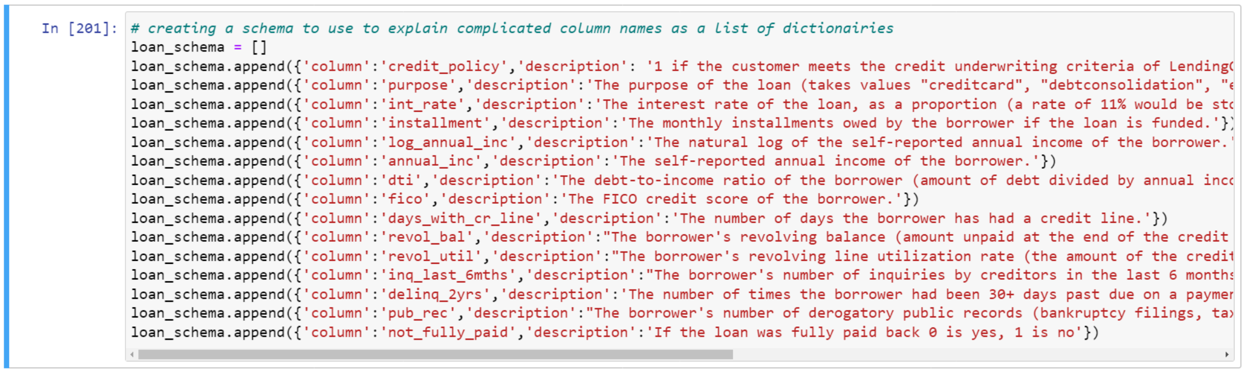
```
In [181]: import numpy as np
          annual_inc = round(np.exp(loanDF['log.annual.inc']),2)
          loanDF.insert(5, 'annual_inc', annual_inc)
```

The column names needed to be fixed. A list of the column names was created and set to replace the existing columns so Python would not have any errors in execution due to the "." being the separator and replace with "_".

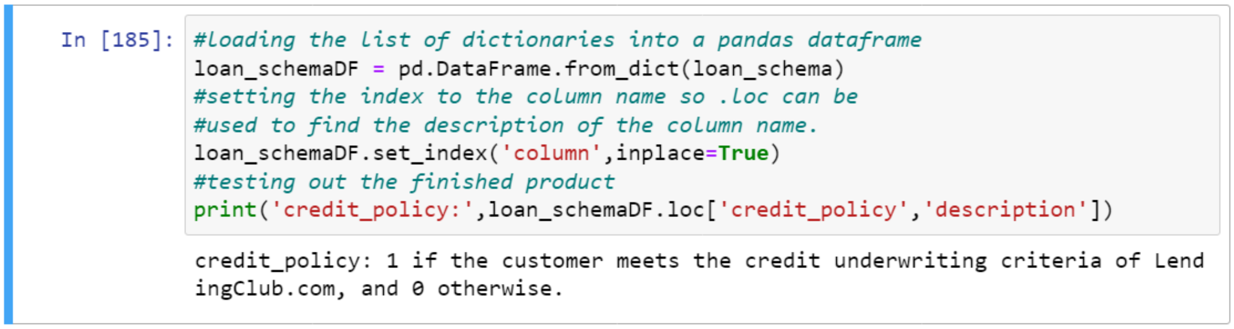


```
In [298]: #replacing the column names
          loanDF.columns = ['credit_policy','purpose', 'interest_rate', 'installment',
                            'log_annual_inc','annual_inc','dti', 'fico',
                            'days_with_cr_line', 'revol_bal','revol_util',
                           'inq_last_6mths', 'delinq_2yrs', 'pub_rec',
                            'not_fully_paid']
```

Since the column names are somewhat confusing a schema data frame was created so we could use the loc function to print the description of the columns being used to help facilitate understanding. A list was created and filled with dictionaries of a key “column” filled with the column names and a key “description” filled with the column meaning. This was placed into a Pandas data frame and the index was set to the column so the information could be easily accessed. When a column is used it will be preceded by the description for understanding purposes.



```
In [201]: # creating a schema to use to explain complicated column names as a list of dictionairies
loan_schema = []
loan_schema.append({'column':'credit_policy','description': '1 if the customer meets the credit underwriting criteria of Lending(
loan_schema.append({'column':'purpose','description':'The purpose of the loan (takes values "creditcard", "debtconsolidation", "e
loan_schema.append({'column':'int_rate','description':'The interest rate of the loan, as a proportion (a rate of 11% would be sto
loan_schema.append({'column':'installment','description':'The monthly installments owed by the borrower if the loan is funded.'})
loan_schema.append({'column':'log_annual_inc','description':'The natural log of the self-reported annual income of the borrower.'
loan_schema.append({'column':'annual_inc','description':'The self-reported annual income of the borrower.'})
loan_schema.append({'column':'dti','description':'The debt-to-income ratio of the borrower (amount of debt divided by annual inco
loan_schema.append({'column':'fico','description':'The FICO credit score of the borrower.'})
loan_schema.append({'column':'days_with_cr_line','description':'The number of days the borrower has had a credit line.'})
loan_schema.append({'column':'revol_bal','description':"The borrower's revolving balance (amount unpaid at the end of the credit
loan_schema.append({'column':'revol_util','description':"The borrower's revolving line utilization rate (the amount of the credit
loan_schema.append({'column':'inq_last_6mths','description':"The borrower's number of inquiries by creditors in the last 6 months
loan_schema.append({'column':'delinq_2yrs','description':'The number of times the borrower had been 30+ days past due on a paymer
loan_schema.append({'column':'pub_rec','description':"The borrower's number of derogatory public records (bankruptcy filings, tax
loan_schema.append({'column':'not_fully_paid','description':'If the loan was fully paid back 0 is yes, 1 is no'})
```



```
In [185]: #Loading the list of dictionaries into a pandas dataframe
loan_schemaDF = pd.DataFrame.from_dict(loan_schema)
#setting the index to the column name so .loc can be
#used to find the description of the column name.
loan_schemaDF.set_index('column',inplace=True)
#testing out the finished product
print('credit_policy:',loan_schemaDF.loc['credit_policy','description'])
```

```
credit_policy: 1 if the customer meets the credit underwriting criteria of Lend
ingClub.com, and 0 otherwise.
```

Reading and cleaning the data (Tweets)

Using snscrape an empty list was created. Get_items() was used with a break point of 10,000. Tweets were gathered and a dictionary of each tweet was placed into a list.

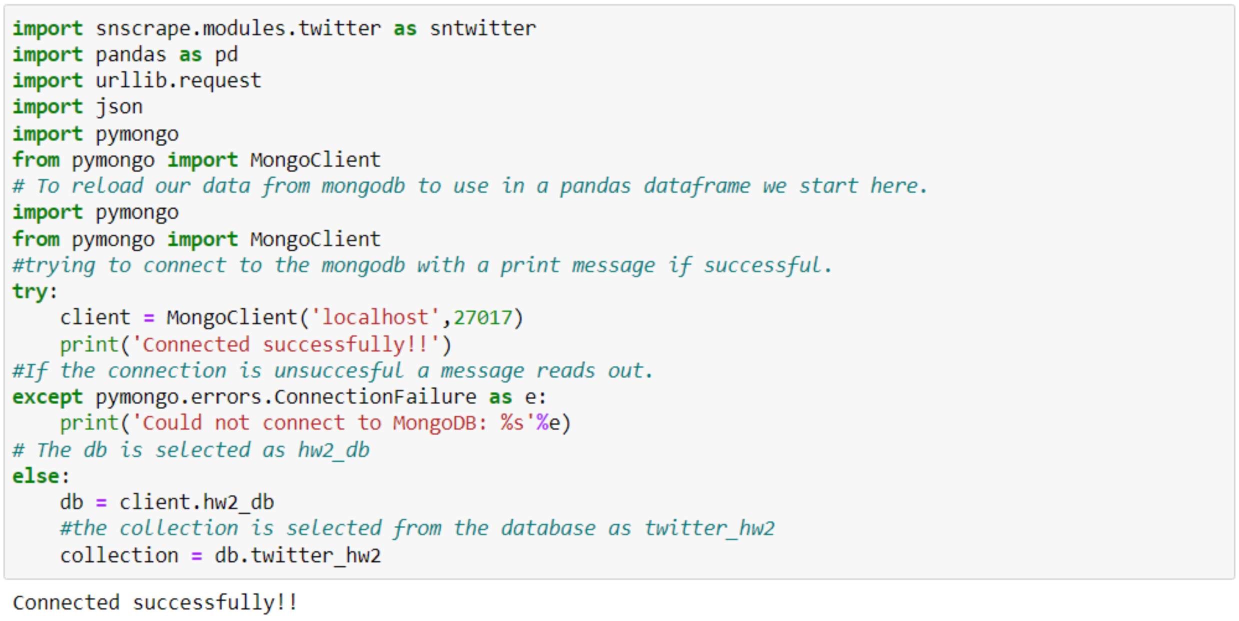
```python
import snscrape.modules.twitter as sntwitter
import pandas as pd
import urllib.request
import json
import pymongo
from pymongo import MongoClient



# Creating an empty list for adding the twiter data into
hw_tweets =[]
# setting i to zero so we can use it as a break point.
i = 0
# using .get_items() to to pull all twitter data for our selected topic.
for tweet in sntwitter.TwitterSearchScraper("dunkin dounts").get_items():
    if i>10000: #selecting a number where when i is greater than it stops the loop.
        break
    # selecting the field we want to pull in from the data and placing them in a dictionary
    # placing that created dictionary into the empty list created at the start.
    hw_tweets.append({'id': tweet.id,
                      'created':tweet.date.strftime("%m/%d/%Y, %H:%M:%S"),
                      'is_reply':tweet.inReplyToTweetId,
                      'content':tweet.content,
                      'username':tweet.user.username,
                      'followers':tweet.user.followersCount,
                      'friends': tweet.user.friendsCount,
                      'device_type':tweet.sourceLabel,
                      'outs':tweet.tcooutlinks,'quoteCount':tweet.quoteCount,
                      'likeCount':tweet.likeCount
                     })
    i= i + 1 #each loop preformed adds one to i.
```

The list of dictionaries, one tweet per dictionary, was saved into mongo data base so the same tweets could be used for the process.

```python
#trying to connect to the mongodb with a print message if successful.
try:
    client = MongoClient('localhost',27017)
    print('Connected successfully!!')
    # used to drop the database when writing the code
    #so it didnt have duplicates/hundreds of thousands of rows
    client.drop_database('hw2_db')
#If the connection is unsuccesful a message reads out.
except pymongo.errors.ConnectionFailure as e:
    print('Could not connect to MongoDB: %s'%e)
# If the connection was established creates a database called hw2_db
else:
    db = client.hw2_db
    # The collection is called twitter_hw2 and saved as collection for use.
    collection = db.twitter_hw2
    #finally many dictioaries are added from our dictionary into the twitter_hw2 collection
    collection.insert_many(hw_tweets)
    # a message prints with the number of rows added to the mongodb
    print('Added', len(hw_tweets),'to tweets collection in hw_two database')
```

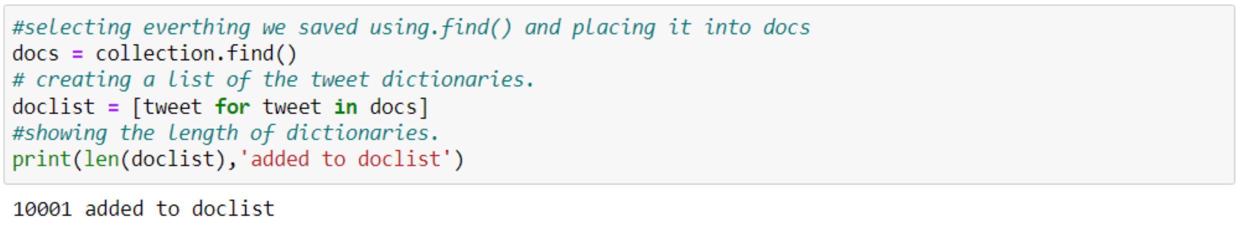
```
Connected successfully!!
Added 10001 to tweets collection in hw_two database
```

The data was pulled back out from the mongo database and placed in into a list of dictionaries.



```
import snscrape.modules.twitter as sntwitter
import pandas as pd
import urllib.request
import json
import pymongo
from pymongo import MongoClient
# To reload our data from mongodb to use in a pandas dataframe we start here.
import pymongo
from pymongo import MongoClient
#trying to connect to the mongodb with a print message if successful.
try:
    client = MongoClient('localhost',27017)
    print('Connected successfully!!')
#If the connection is unsuccesful a message reads out.
except pymongo.errors.ConnectionFailure as e:
    print('Could not connect to MongoDB: %s'%e)
# The db is selected as hw2_db
else:
    db = client.hw2_db
    #the collection is selected from the database as twitter_hw2
    collection = db.twitter_hw2
```

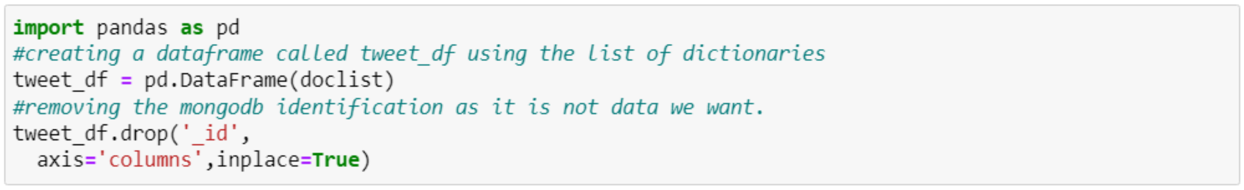
```
Connected successfully!!
```



```
#selecting everthing we saved using.find() and placing it into docs
docs = collection.find()
# creating a list of the tweet dictionaries.
doclist = [tweet for tweet in docs]
#showing the length of dictionaries.
print(len(doclist),'added to doclist')
```

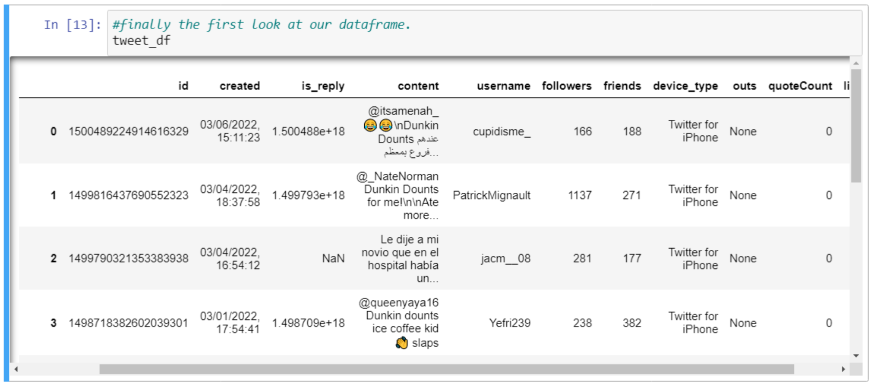
```
10001 added to doclist
```

The list was converted into a pandas data frame and the mongo database id was dropped as it isn't information needed for analysis.



```
import pandas as pd
#creating a dataframe called tweet_df using the list of dictionaries
tweet_df = pd.DataFrame(doclist)
#removing the mongodb identification as it is not data we want.
tweet_df.drop('_id',
  axis='columns',inplace=True)
```

The first look at the pandas data frame containing 10,001 tweets!

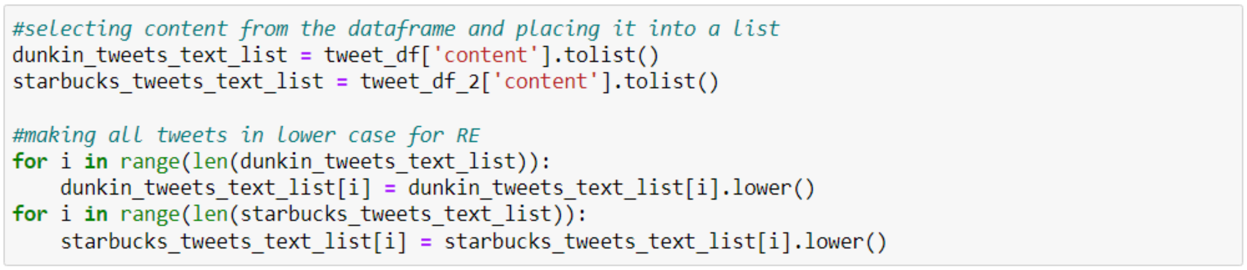


```
In [13]: #finally the first look at our dataframe.
         tweet_df
```

	id	created	is_reply	content	username	followers	friends	device_type	outs	quoteCount	li
0	1500489224914616329	03/06/2022, 15:11:23	1.500488e+18	@itsamenah_ 😂😂\nDunkin Dounts عندهم فروع بمعظم...	cupidisme_	166	188	Twitter for iPhone	None	0	
1	1499816437690552323	03/04/2022, 18:37:58	1.499793e+18	@_NateNorman Dunkin Dounts for me!\n\nAte more...	PatrickMignault	1137	271	Twitter for iPhone	None	0	
2	1499790321353383938	03/04/2022, 16:54:12	NaN	Le dije a mi novio que en el hospital había un...	jacm__08	281	177	Twitter for iPhone	None	0	
3	1498718382602039301	03/01/2022, 17:54:41	1.498709e+18	@queenyaya16 Dunkin dounts ice coffee kid 👏 slaps	Yefri239	238	382	Twitter for iPhone	None	0	

This process was repeated replacing 'Dunkin Donuts' with 'Starbucks' and 'Toys-R-Us'.

The content column, which contains the actual tweet, was selected then stored into a list. The lists were run through a loop to change all the characters to lower case letters.



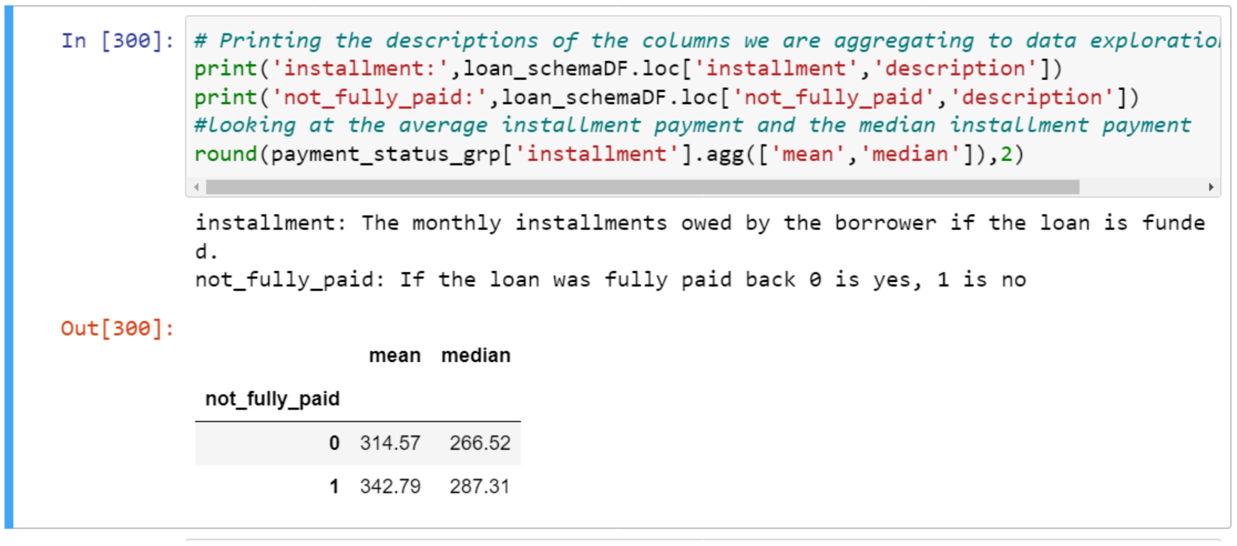
```
#selecting content from the dataframe and placing it into a list
dunkin_tweets_text_list = tweet_df['content'].tolist()
starbucks_tweets_text_list = tweet_df_2['content'].tolist()

#making all tweets in lower case for RE
for i in range(len(dunkin_tweets_text_list)):
    dunkin_tweets_text_list[i] = dunkin_tweets_text_list[i].lower()
for i in range(len(starbucks_tweets_text_list)):
    starbucks_tweets_text_list[i] = starbucks_tweets_text_list[i].lower()
```

This process was repeated with the Toys-R-Us text list.

Analysis (Kaggle loan Data set)

Question #1 - How does the average monthly amount due change in loans that were paid back vs loans that were not paid?

Using the created aggregated data frame, the 'installment' column is specified, and the aggregate function is used passing in mean and median to see the monthly average installments for each grouping.   The Median is also shown to see if one grouping has a larger amount of skewing over the other.



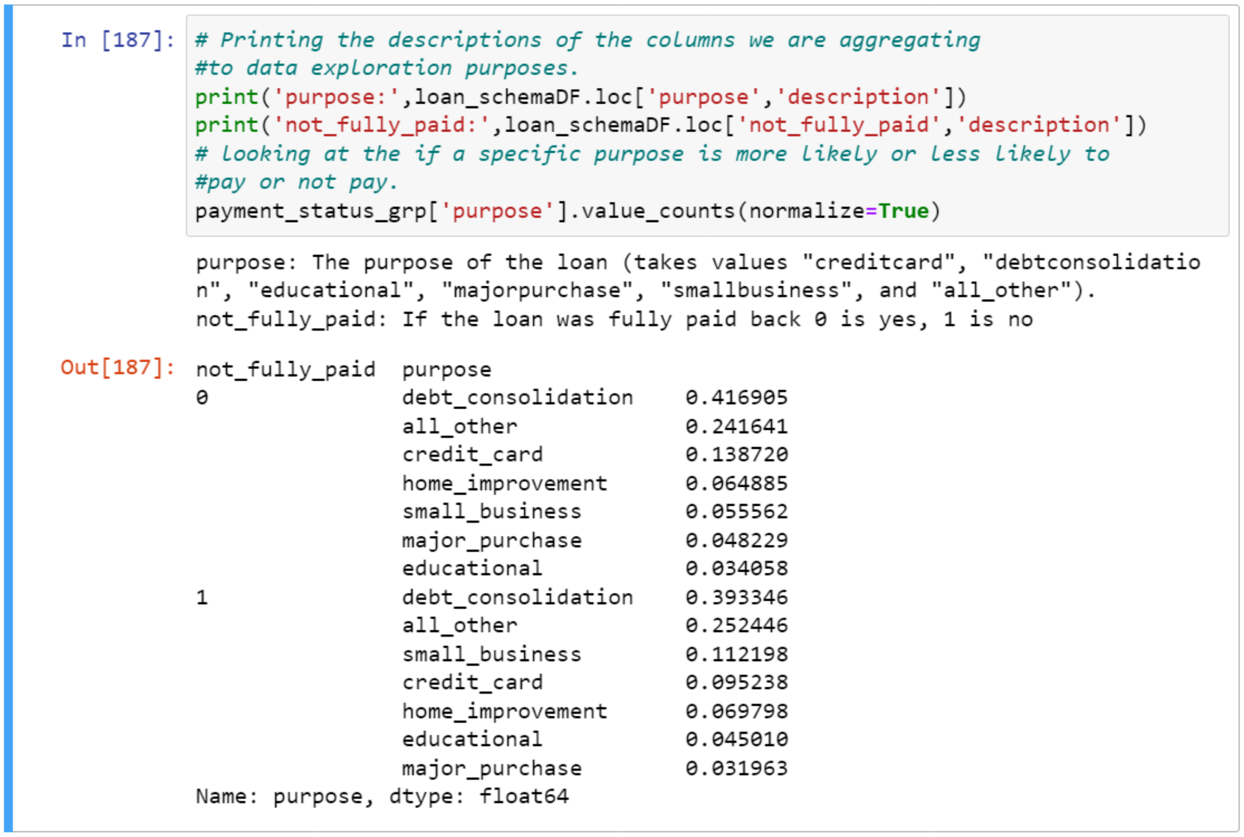
```
In [300]: # Printing the descriptions of the columns we are aggregating to data exploratio
          print('installment:',loan_schemaDF.loc['installment','description'])
          print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
          #Looking at the average installment payment and the median installment payment
          round(payment_status_grp['installment'].agg(['mean','median']),2)

          installment: The monthly installments owed by the borrower if the loan is funde
          d.
          not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Out[300]:
```

	mean	median
not_fully_paid		
0	314.57	266.52
1	342.79	287.31

We can see that the average installment needed to be paid is $28 (which is about an 8%) more in loans that defaulted. This could be a significant amount difference depending on who is paying this loan.

Question #2 - Does the purpose of the loan change the rate of being paid back or not?

Using the created aggregated data frame, the ‘purpose’ column is specified and the value_counts function is used passing in normalize=True so the results are shown as percentages.

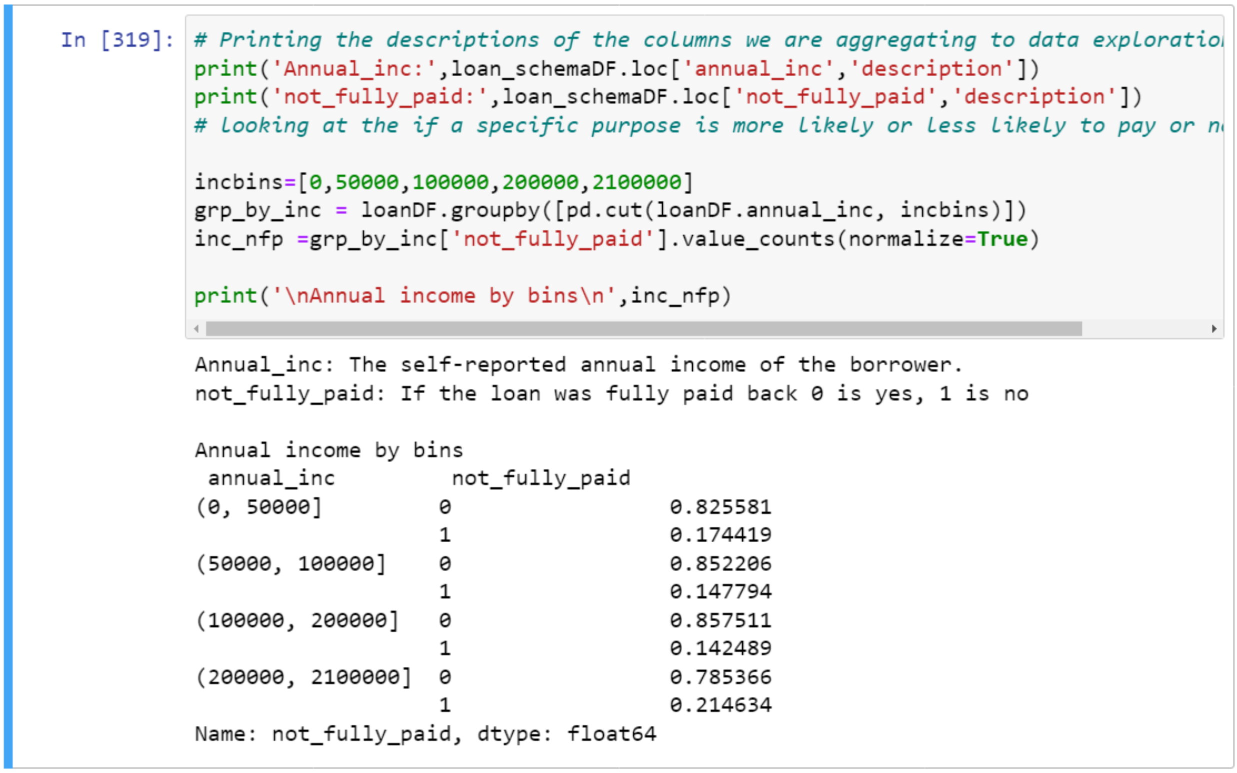


```
In [187]: # Printing the descriptions of the columns we are aggregating
          #to data exploration purposes.
          print('purpose:',loan_schemaDF.loc['purpose','description'])
          print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
          # Looking at the if a specific purpose is more likely or less likely to
          #pay or not pay.
          payment_status_grp['purpose'].value_counts(normalize=True)

          purpose: The purpose of the loan (takes values "creditcard", "debtconsolidatio
          n", "educational", "majorpurchase", "smallbusiness", and "all_other").
          not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Out[187]: not_fully_paid  purpose
          0               debt_consolidation    0.416905
                          all_other             0.241641
                          credit_card           0.138720
                          home_improvement      0.064885
                          small_business        0.055562
                          major_purchase        0.048229
                          educational           0.034058
          1               debt_consolidation    0.393346
                          all_other             0.252446
                          small_business        0.112198
                          credit_card           0.095238
                          home_improvement      0.069798
                          educational           0.045010
                          major_purchase        0.031963
          Name: purpose, dtype: float64
```

This shows that small business loans are a larger percent of being not paid vs the percent of them being paid back. This could let the loan lending club know that they should be more stringent on small business loans.

Question #3a - Does annual income change the percentage of loans being paid back?

Income bins were created ranging from $0-$50,000, $50,001-$100,000, $100,001-$200,000, and $200,001-$2,100,000. The last bin represents the rows with an annual income of over $200,000. These bins were used when to group the original data frame by the annual_inc column. The not_fully_paid column was selected and the value_counts function was used this time with normalize=True to see the percentage of rows that paid back in full vs not paid fully for each bin created.



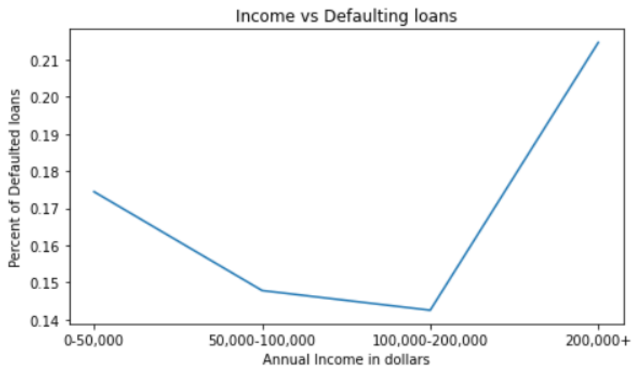
```
In [319]: # Printing the descriptions of the columns we are aggregating to data exploratio
          print('Annual_inc:',loan_schemaDF.loc['annual_inc','description'])
          print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
          # looking at the if a specific purpose is more likely or less likely to pay or n

          incbins=[0,50000,100000,200000,2100000]
          grp_by_inc = loanDF.groupby([pd.cut(loanDF.annual_inc, incbins)])
          inc_nfp =grp_by_inc['not_fully_paid'].value_counts(normalize=True)

          print('\nAnnual income by bins\n',inc_nfp)
```

```
Annual_inc: The self-reported annual income of the borrower.
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Annual income by bins
 annual_inc         not_fully_paid
(0, 50000]         0                 0.825581
                   1                 0.174419
(50000, 100000]    0                 0.852206
                   1                 0.147794
(100000, 200000]   0                 0.857511
                   1                 0.142489
(200000, 2100000]  0                 0.785366
                   1                 0.214634
Name: not_fully_paid, dtype: float64
```

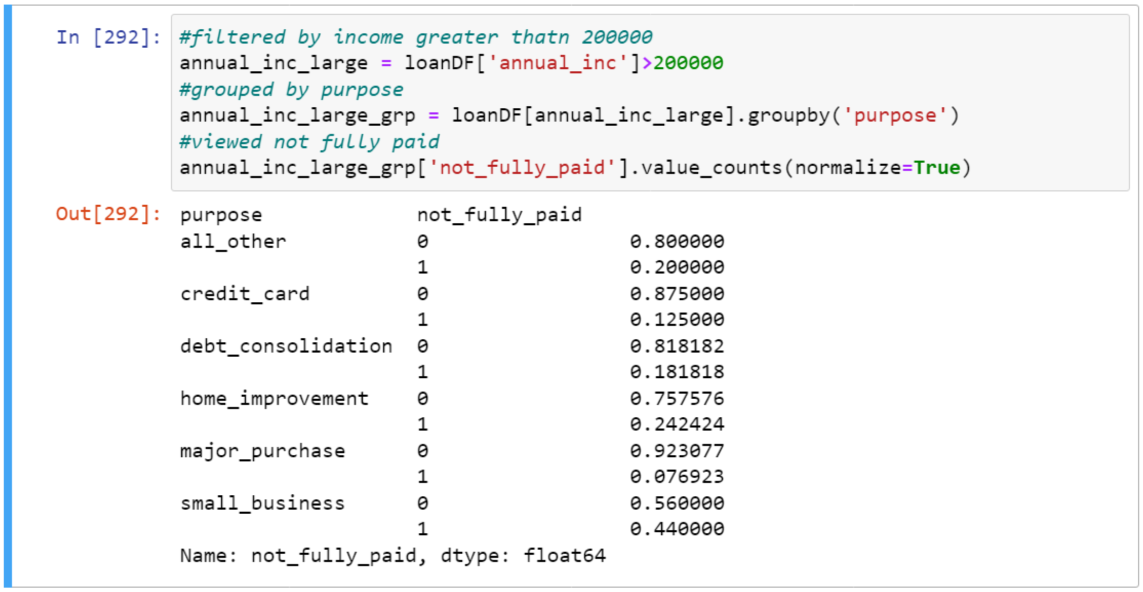
At first it is as expected as the amount of annual income increases the chance the loan is paid back increases until the income overtakes $200,000.

A line graph was created to show the Defaulted loan percentages.

```
In [366]: names = ['0-50,000', '50,000-100,000', '100,000-200,000','200,000+']
          values = [inc_nfp[1],inc_nfp[3],inc_nfp[5],inc_nfp[7]]
          plt.figure(figsize=(25,4))

          plt.subplot(131)
          plt.plot(names, values)
          plt.ylabel('Percent of Defaulted loans')
          plt.xlabel('Annual Income in dollars')
          plt.title('Income vs Defaulting loans')
          plt.show()
```



To understand why this is occurring a new variable is created containing the rows of the data frame where the annual_inc is greater than $200,000, this filtered data frame is grouped by the purpose column to see what loans are not being paid back.



```
In [292]: #filtered by income greater thatn 200000
          annual_inc_large = loanDF['annual_inc']>200000
          #grouped by purpose
          annual_inc_large_grp = loanDF[annual_inc_large].groupby('purpose')
          #viewed not fully paid
          annual_inc_large_grp['not_fully_paid'].value_counts(normalize=True)

Out[292]: purpose             not_fully_paid
          all_other           0                 0.800000
                              1                 0.200000
          credit_card         0                 0.875000
                              1                 0.125000
          debt_consolidation  0                 0.818182
                              1                 0.181818
          home_improvement    0                 0.757576
                              1                 0.242424
          major_purchase      0                 0.923077
                              1                 0.076923
          small_business      0                 0.560000
                              1                 0.440000
          Name: not_fully_paid, dtype: float64
```

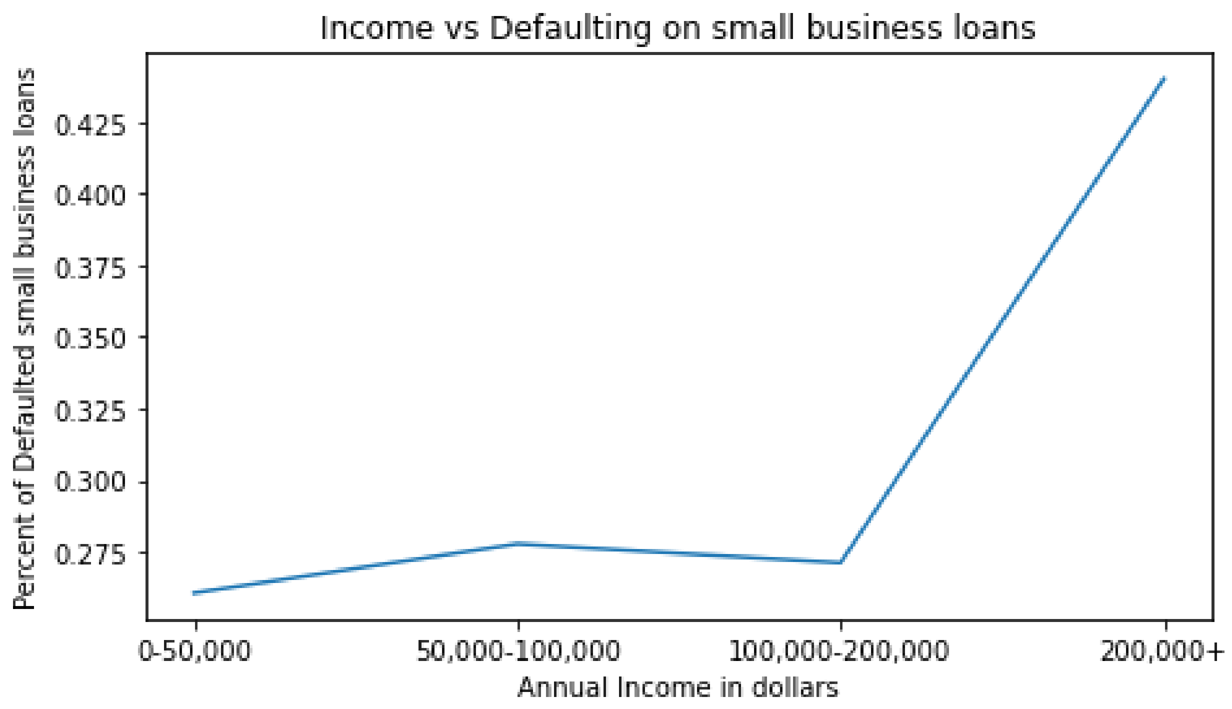
As opposed to the when looking at all the data grouped by purpose of the loans, these are being paid back at much higher rates in each type compared to the whole of the data set. This is not true of the small business loans. It was shown earlier that small business loans have a higher rate of not being paid back and this seems to further support that in a much higher amount.

Question #3b - *What annual income ranges produces the least amount of defaulted small business loans?

A new filtered data frame was created that only contained small business loans. This was then grouped by annual_inc and value_counts was used with normalize = True to find the percentage of small business loans that were paid back vs not for each income bin created.

```python
# Printing the descriptions of the columns we are aggregating to data exploration purposes.
print('Annual_inc:',loan_schemaDF.loc['annual_inc','description'])
print('purpose:',loan_schemaDF.loc['not_fully_paid','description'])
print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
# looking at the if a specific purpose is more likely or less likely to pay or not pay.

#Selecting only small business loans
sb= loanDF.loc[loanDF['purpose'] == 'small_business']

# selecting the income ranges
incbins=[0,50000,100000,200000,2100000]
# Grouping the small buisiness loans by income
sb_grp_by_inc = sb.groupby([pd.cut(sb.annual_inc, incbins)])
# Creating a % of paid vs not paid
sb_inc_nfp =sb_grp_by_inc['not_fully_paid'].value_counts(normalize=True)

print('\nAnnual income by bins\n',sb_inc_nfp)
```

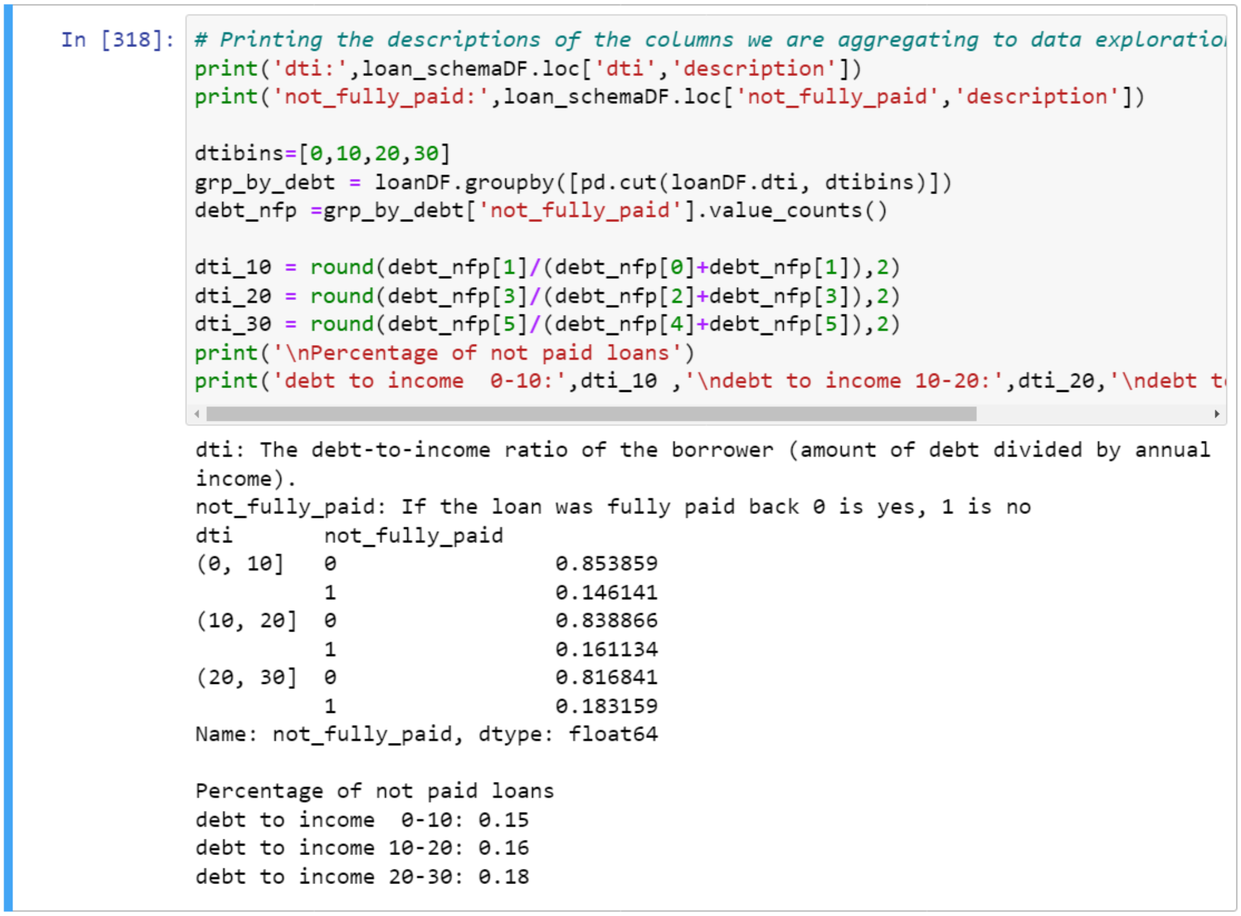
```
Annual_inc: The self-reported annual income of the borrower.
purpose: If the loan was fully paid back 0 is yes, 1 is no
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Annual income by bins
 annual_inc         not_fully_paid
(0, 50000]         0                 0.739362
                   1                 0.260638
(50000, 100000]    0                 0.722222
                   1                 0.277778
(100000, 200000]   0                 0.728814
                   1                 0.271186
(200000, 2100000]  0                 0.560000
                   1                 0.440000
Name: not_fully_paid, dtype: float64
```



This shows that small business loans are defaulted at a higher rate when reported income is greater than $200,000.

Question #4a - Does more debt-to-income ratio change the percentage of loans being paid back?

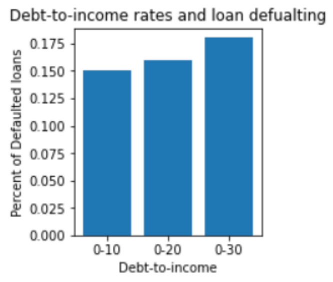
Bins were created representing debt-to-income (dti) from 0-10, 11-20, and 21-30. The original data frame was grouped by the dti column using these bins. This grouped by data frame was specified to look at not_fully_paid and the value-counts. To try something new instead of using the normalize= True the counts were selected for and the percentage of loans not paid back was displaced.



```
In [318]: # Printing the descriptions of the columns we are aggregating to data exploratio
          print('dti:',loan_schemaDF.loc['dti','description'])
          print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])

          dtibins=[0,10,20,30]
          grp_by_debt = loanDF.groupby([pd.cut(loanDF.dti, dtibins)])
          debt_nfp =grp_by_debt['not_fully_paid'].value_counts()

          dti_10 = round(debt_nfp[1]/(debt_nfp[0]+debt_nfp[1]),2)
          dti_20 = round(debt_nfp[3]/(debt_nfp[2]+debt_nfp[3]),2)
          dti_30 = round(debt_nfp[5]/(debt_nfp[4]+debt_nfp[5]),2)
          print('\nPercentage of not paid loans')
          print('debt to income  0-10:',dti_10 ,'\ndebt to income 10-20:',dti_20,'\ndebt t

          dti: The debt-to-income ratio of the borrower (amount of debt divided by annual
          income).
          not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no
          dti       not_fully_paid
          (0, 10]   0                 0.853859
                    1                 0.146141
          (10, 20]  0                 0.838866
                    1                 0.161134
          (20, 30]  0                 0.816841
                    1                 0.183159
          Name: not_fully_paid, dtype: float64

          Percentage of not paid loans
          debt to income  0-10: 0.15
          debt to income 10-20: 0.16
          debt to income 20-30: 0.18
```

A graph was created to show the Defaulting loan increase as Debt-to-income increases.

```
In [353]: import matplotlib.pyplot as plt

          names = ['0-10',
                   '0-20',
                   '0-30']
          values = [dti_10,dti_20,dti_30]

          plt.figure(figsize=(9,3))

          plt.subplot(131)
          plt.bar(names, values)
          plt.ylabel('Percent of Defaulted loans')
          plt.xlabel('Debt-to-income')
          plt.title('Debt-to-income rates and loan defualting')
          plt.show()
```



As the debt-to-income increases so does the chance the loan will not be paid back in full.

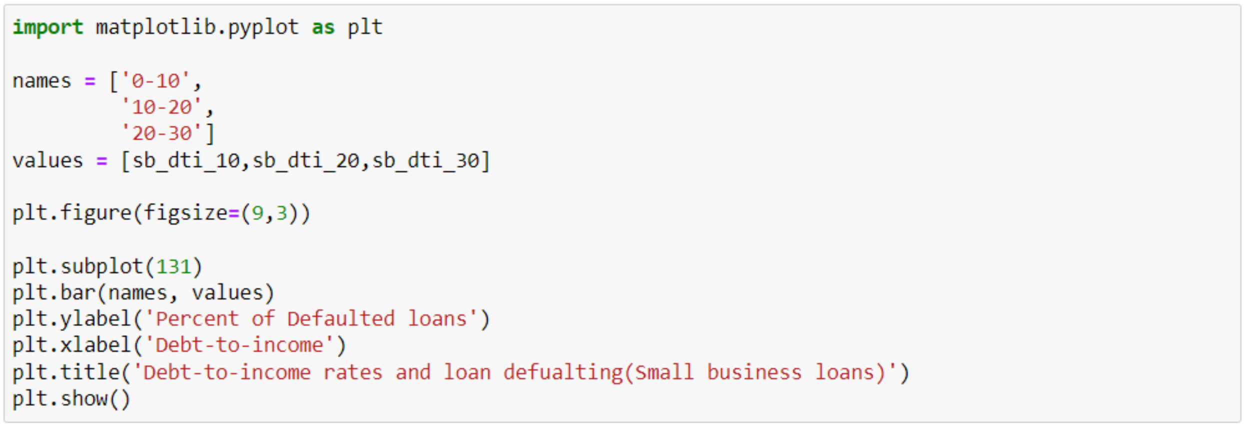
Question #4b - *How does debt-to-income ratio change with small business loans?

The previously created small business loan data frame was grouped by dti (debt-to-income) column using the same dti (debt-to-income) bins.

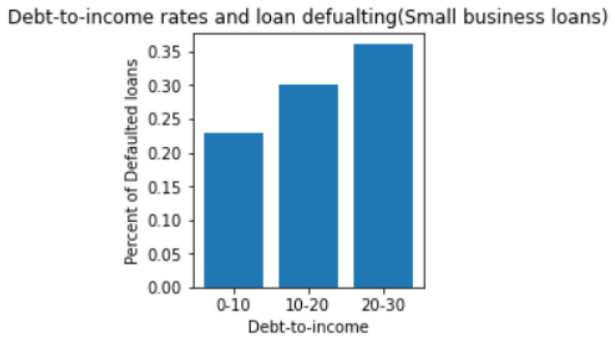
```
# Printing the descriptions of the columns we are aggregating to data exploration purposes.
print('dti:',loan_schemaDF.loc['dti','description'])
print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
print('purpose:',loan_schemaDF.loc['not_fully_paid','description'])

dtibins=[0,10,20,30]
sb_grp_by_debt = sb.groupby([pd.cut(sb.dti, dtibins)])
sb_debt_nfp =sb_grp_by_debt['not_fully_paid'].value_counts()

sb_dti_10 = round(sb_debt_nfp[1]/(sb_debt_nfp[0]+sb_debt_nfp[1]),2)
sb_dti_20 = round(sb_debt_nfp[3]/(sb_debt_nfp[2]+sb_debt_nfp[3]),2)
sb_dti_30 = round(sb_debt_nfp[5]/(sb_debt_nfp[4]+sb_debt_nfp[5]),2)
print('\nPercentage of not paid small business loans')
print('debt to income  0-10:',sb_dti_10 ,'\ndebt to income 10-20:',sb_dti_20,'\ndebt to income 20-30:',
```

```
dti: The debt-to-income ratio of the borrower (amount of debt divided by annual income).
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no
purpose: If the loan was fully paid back 0 is yes, 1 is no

Percentage of not paid small business loans
debt to income  0-10: 0.23
debt to income 10-20: 0.3
debt to income 20-30: 0.36
```

A graph was created to show the Defaulting loan increase as Debt-to-income increases.



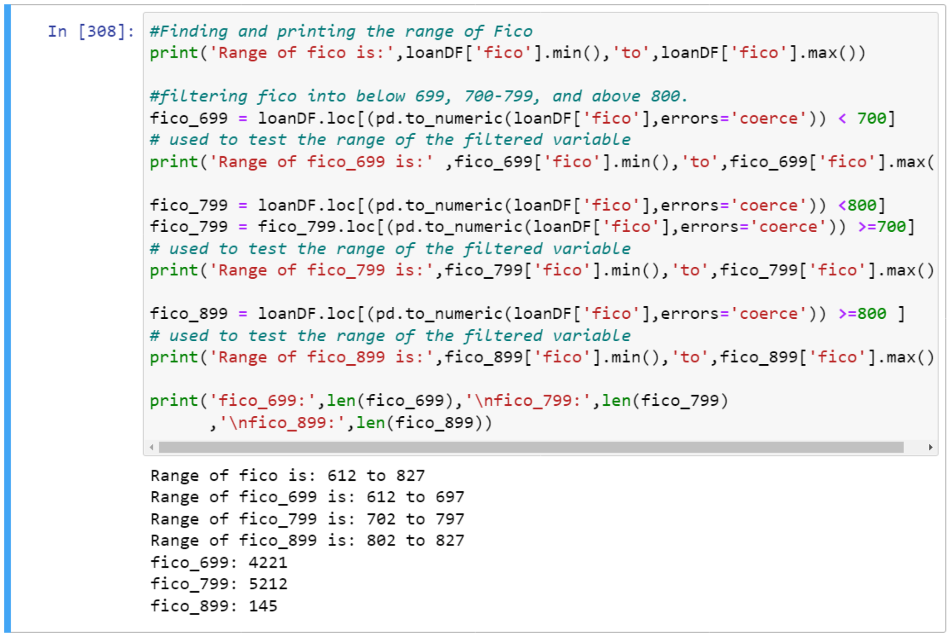
```
import matplotlib.pyplot as plt

names = ['0-10',
         '10-20',
         '20-30']
values = [sb_dti_10,sb_dti_20,sb_dti_30]

plt.figure(figsize=(9,3))

plt.subplot(131)
plt.bar(names, values)
plt.ylabel('Percent of Defaulted loans')
plt.xlabel('Debt-to-income')
plt.title('Debt-to-income rates and loan defualting(Small business loans)')
plt.show()
```



This graph shows us that as a borrower has a higher debt-to-income ratio they default at a higher chance to default. When comparing the two graphs there is a higher percentage of defulters in each dti (debt-to-income) section, but the increase is much more severe as the dti increases with small business loans.

<u>Question #5a</u> - How does fico credit score change the percentage of loans being paid vs not?

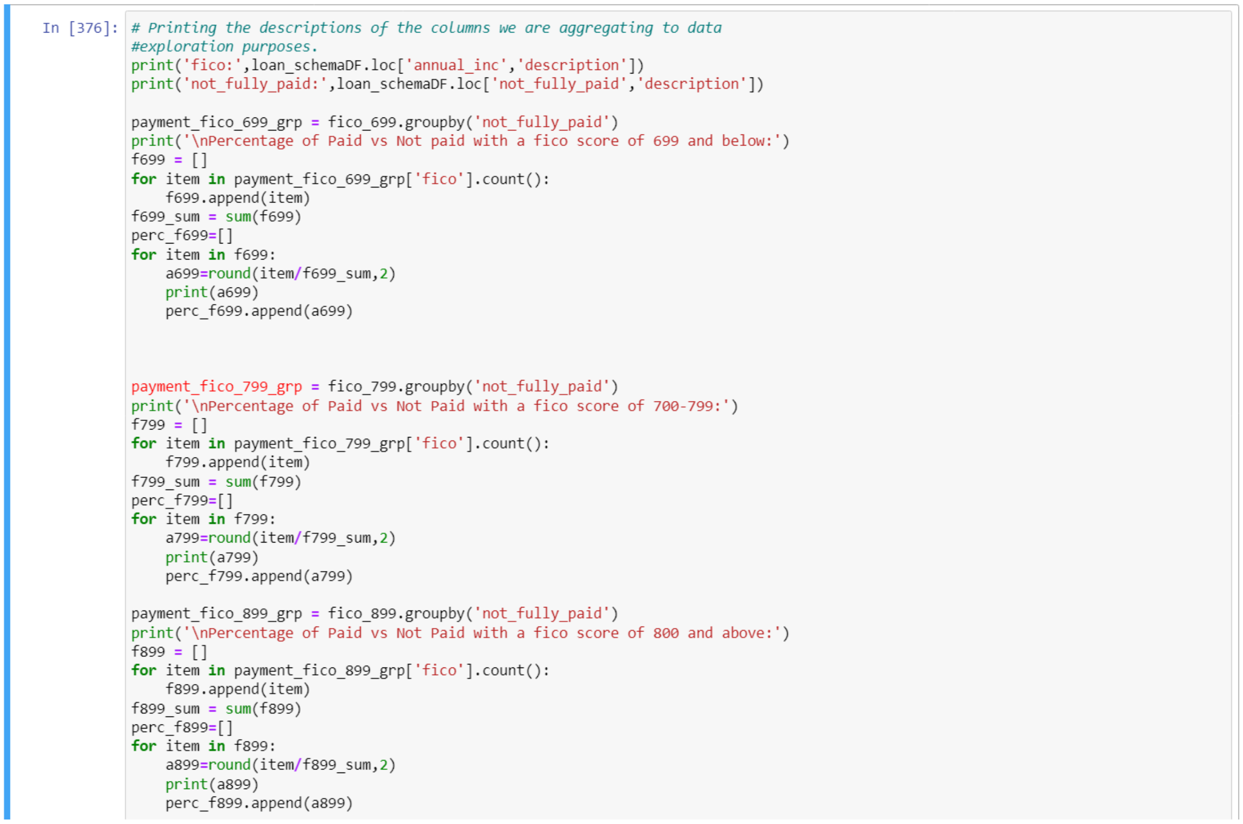
This original data frame selecting the fico column was used with the min and max function to find a range of the credit scores. Three variables were created having a section of the data frame one containing credit score below 699 (fico_699), one with credit scores from 700 – 799 (fico_799) and lastly one with above 800 (fico_899). This was done using filtering and Boolean operators. The range is tested and printed out along with the number of rows using the length function. This is significant as the 800 and above has a much smaller number of rows and this data may not be accurate because of our small sample size. Although it could be argued that this small number is a good representation of the population because it is hard to acquire a fico credit score above 800.



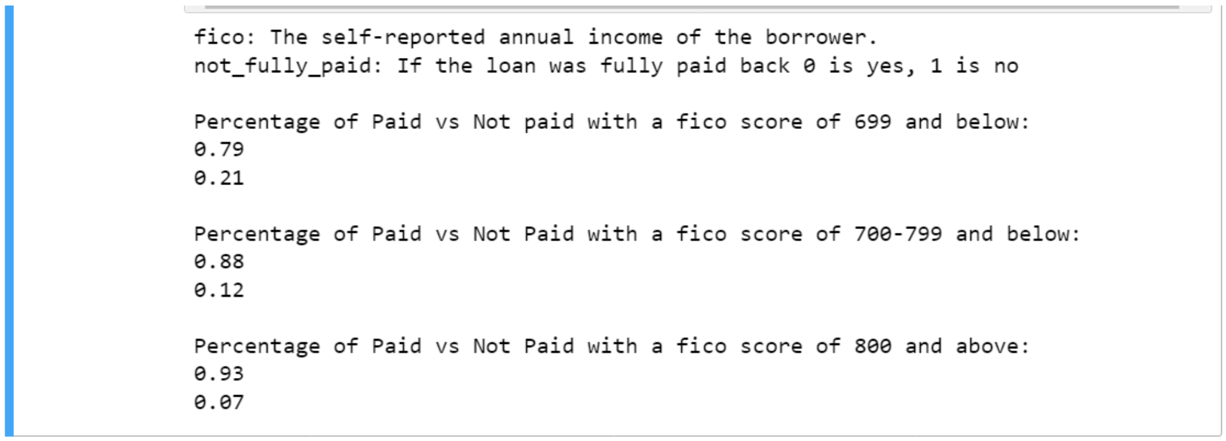
```
In [308]: #Finding and printing the range of Fico
          print('Range of fico is:',loanDF['fico'].min(),'to',loanDF['fico'].max())

          #filtering fico into below 699, 700-799, and above 800.
          fico_699 = loanDF.loc[(pd.to_numeric(loanDF['fico'],errors='coerce')) < 700]
          # used to test the range of the filtered variable
          print('Range of fico_699 is:' ,fico_699['fico'].min(),'to',fico_699['fico'].max(

          fico_799 = loanDF.loc[(pd.to_numeric(loanDF['fico'],errors='coerce')) <800]
          fico_799 = fico_799.loc[(pd.to_numeric(loanDF['fico'],errors='coerce')) >=700]
          # used to test the range of the filtered variable
          print('Range of fico_799 is:',fico_799['fico'].min(),'to',fico_799['fico'].max()

          fico_899 = loanDF.loc[(pd.to_numeric(loanDF['fico'],errors='coerce')) >=800 ]
          # used to test the range of the filtered variable
          print('Range of fico_899 is:',fico_899['fico'].min(),'to',fico_899['fico'].max()

          print('fico_699:',len(fico_699),'\nfico_799:',len(fico_799)
                ,'\nfico_899:',len(fico_899))
```

```
Range of fico is: 612 to 827
Range of fico_699 is: 612 to 697
Range of fico_799 is: 702 to 797
Range of fico_899 is: 802 to 827
fico_699: 4221
fico_799: 5212
fico_899: 145
```

Each of these filtered variables were then grouped by the not_fully_paid column and run through a loop to find the percent of paid loans vs unpaid loans.

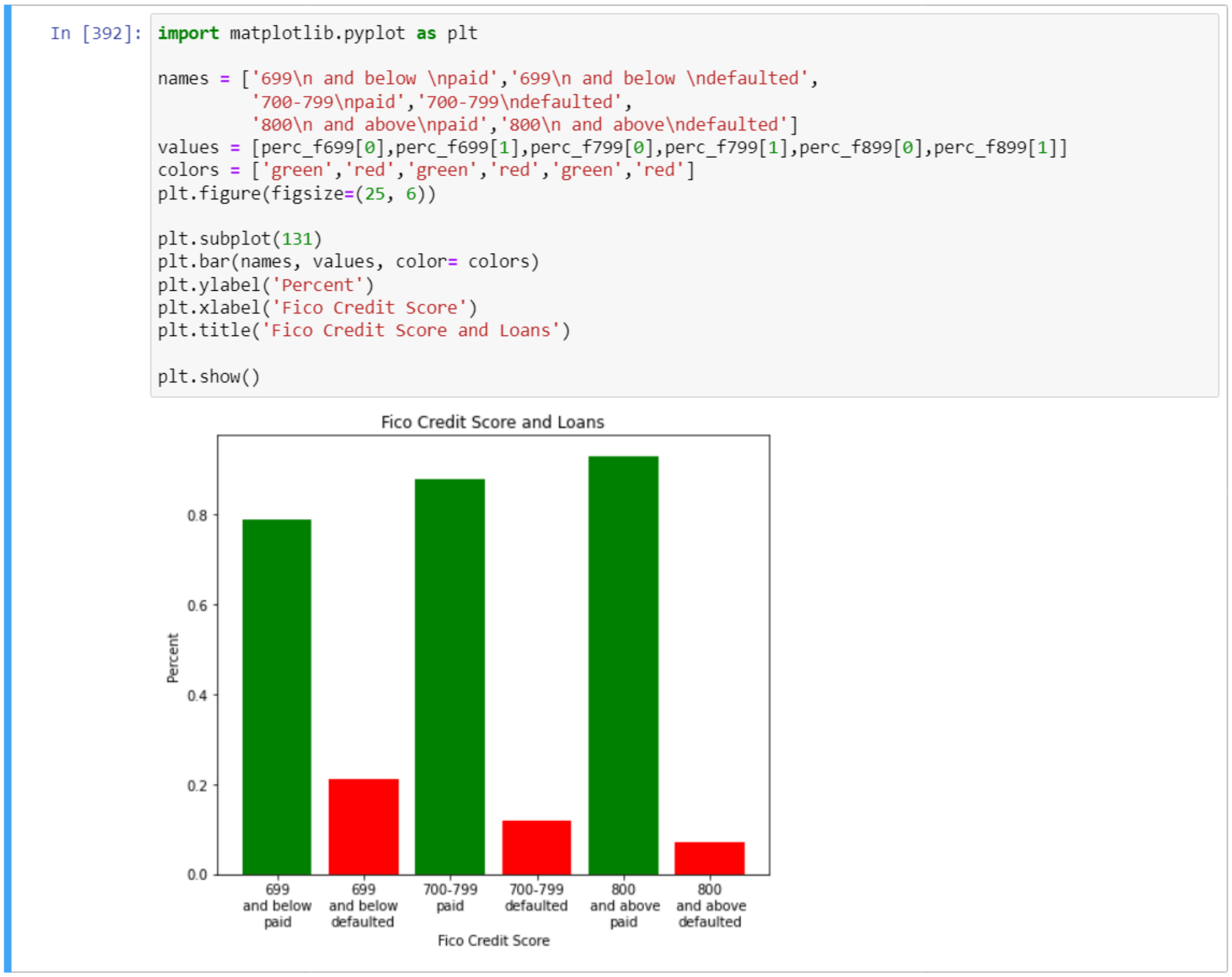


```
In [376]: # Printing the descriptions of the columns we are aggregating to data
          #exploration purposes.
          print('fico:',loan_schemaDF.loc['annual_inc','description'])
          print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])

          payment_fico_699_grp = fico_699.groupby('not_fully_paid')
          print('\nPercentage of Paid vs Not paid with a fico score of 699 and below:')
          f699 = []
          for item in payment_fico_699_grp['fico'].count():
              f699.append(item)
          f699_sum = sum(f699)
          perc_f699=[]
          for item in f699:
              a699=round(item/f699_sum,2)
              print(a699)
              perc_f699.append(a699)



          payment_fico_799_grp = fico_799.groupby('not_fully_paid')
          print('\nPercentage of Paid vs Not Paid with a fico score of 700-799:')
          f799 = []
          for item in payment_fico_799_grp['fico'].count():
              f799.append(item)
          f799_sum = sum(f799)
          perc_f799=[]
          for item in f799:
              a799=round(item/f799_sum,2)
              print(a799)
              perc_f799.append(a799)

          payment_fico_899_grp = fico_899.groupby('not_fully_paid')
          print('\nPercentage of Paid vs Not Paid with a fico score of 800 and above:')
          f899 = []
          for item in payment_fico_899_grp['fico'].count():
              f899.append(item)
          f899_sum = sum(f899)
          perc_f899=[]
          for item in f899:
              a899=round(item/f899_sum,2)
              print(a899)
              perc_f899.append(a899)
```



```
fico: The self-reported annual income of the borrower.
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Percentage of Paid vs Not paid with a fico score of 699 and below:
0.79
0.21

Percentage of Paid vs Not Paid with a fico score of 700-799 and below:
0.88
0.12

Percentage of Paid vs Not Paid with a fico score of 800 and above:
0.93
0.07
```

A graph was created to show the increase in loans paid back and the decrease in Defaulted loans

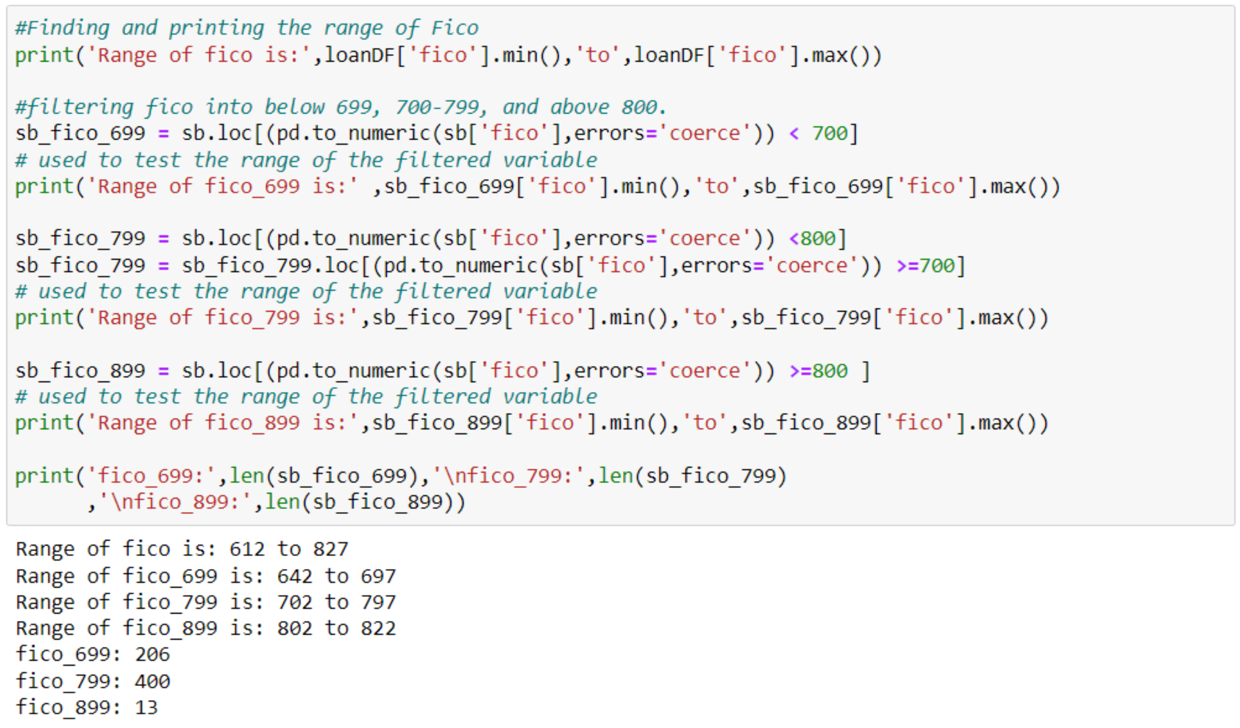


```
In [392]: import matplotlib.pyplot as plt

names = ['699\n and below \npaid','699\n and below \ndefaulted',
         '700-799\npaid','700-799\ndefaulted',
         '800\n and above\npaid','800\n and above\ndefaulted']
values = [perc_f699[0],perc_f699[1],perc_f799[0],perc_f799[1],perc_f899[0],perc_f899[1]]
colors = ['green','red','green','red','green','red']
plt.figure(figsize=(25, 6))

plt.subplot(131)
plt.bar(names, values, color= colors)
plt.ylabel('Percent')
plt.xlabel('Fico Credit Score')
plt.title('Fico Credit Score and Loans')

plt.show()
```

As the fico credit score increases from one bin to the next the percentage of defaulted loans decreases. Showing that credit score is an important indicator of how often a loan defaults.

Queston #5b -*What fico credit score rating range produces the least amount of defaulted small business loans?

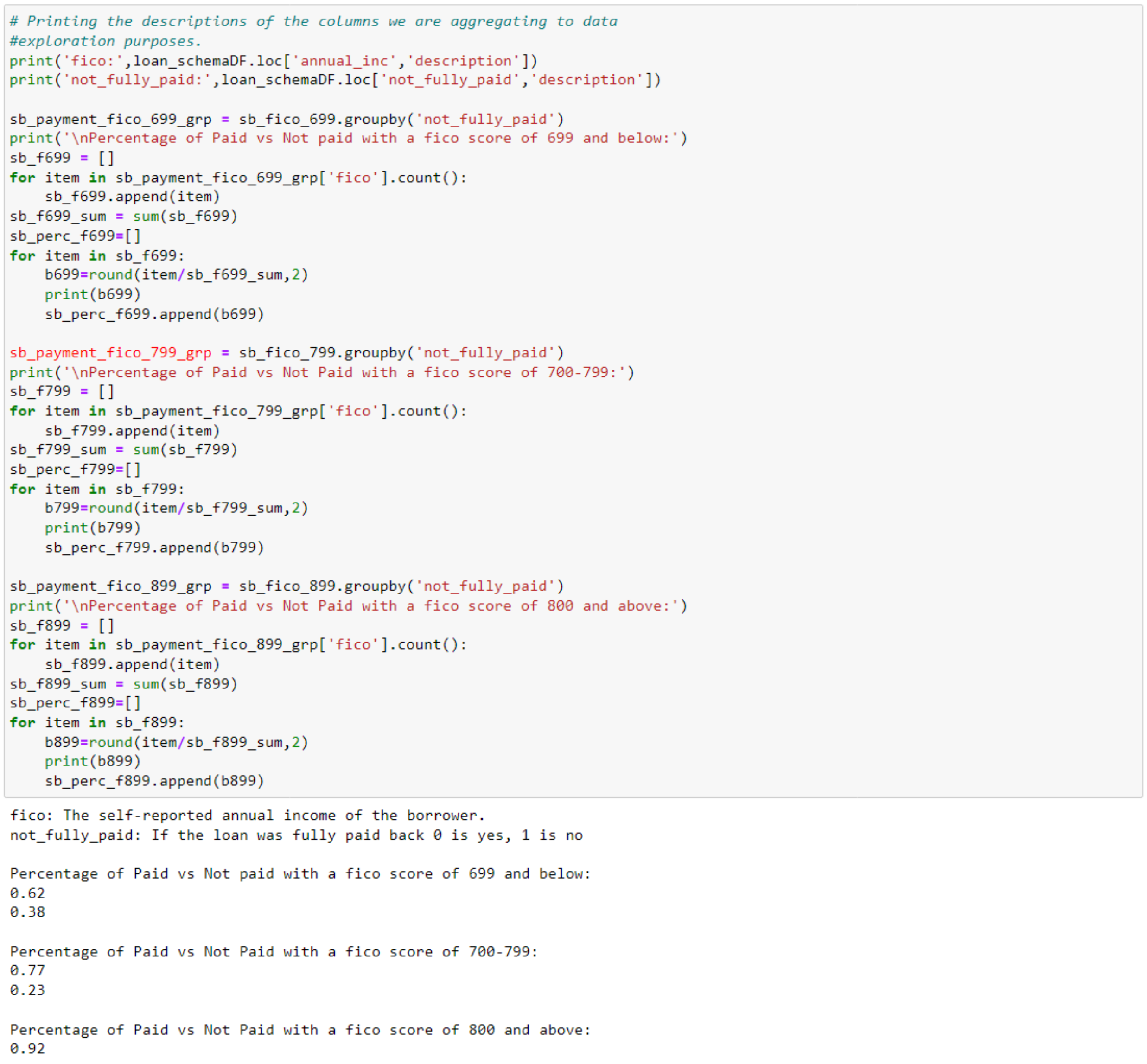
Using the previously created small business loan data frame, the fico column was used with the min and max function to find a range of the credit scores. This was done to make sure the same range existed. Since the range was the same, three variables were created having a section of the small business data frame one containing credit score below 699 (sb_fico_699), one with credit scores from 700 – 799 (sb_fico_799) and lastly one with above 800 (sb_fico_899). This was done using filtering and Boolean operators, same as before. The range is tested and printed out along with the number of rows using the length function. Each fico score rating is similar in size to the original data frame selections.



```
#Finding and printing the range of Fico
print('Range of fico is:',loanDF['fico'].min(),'to',loanDF['fico'].max())

#filtering fico into below 699, 700-799, and above 800.
sb_fico_699 = sb.loc[(pd.to_numeric(sb['fico'],errors='coerce')) < 700]
# used to test the range of the filtered variable
print('Range of fico_699 is:' ,sb_fico_699['fico'].min(),'to',sb_fico_699['fico'].max())

sb_fico_799 = sb.loc[(pd.to_numeric(sb['fico'],errors='coerce')) <800]
sb_fico_799 = sb_fico_799.loc[(pd.to_numeric(sb['fico'],errors='coerce')) >=700]
# used to test the range of the filtered variable
print('Range of fico_799 is:',sb_fico_799['fico'].min(),'to',sb_fico_799['fico'].max())

sb_fico_899 = sb.loc[(pd.to_numeric(sb['fico'],errors='coerce')) >=800 ]
# used to test the range of the filtered variable
print('Range of fico_899 is:',sb_fico_899['fico'].min(),'to',sb_fico_899['fico'].max())

print('fico_699:',len(sb_fico_699),'\nfico_799:',len(sb_fico_799)
      ,'\nfico_899:',len(sb_fico_899))
```

```
Range of fico is: 612 to 827
Range of fico_699 is: 642 to 697
Range of fico_799 is: 702 to 797
Range of fico_899 is: 802 to 822
fico_699: 206
fico_799: 400
fico_899: 13
```

Each of these filtered variables were then grouped by the not_fully_paid column and run through a loop to find the percent of paid loans vs unpaid loans.

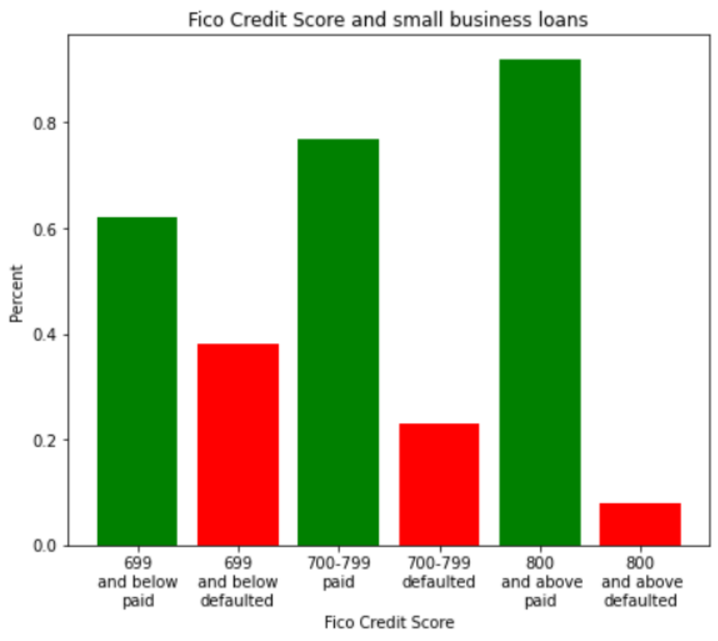


```python
# Printing the descriptions of the columns we are aggregating to data
#exploration purposes.
print('fico:',loan_schemaDF.loc['annual_inc','description'])
print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])

sb_payment_fico_699_grp = sb_fico_699.groupby('not_fully_paid')
print('\nPercentage of Paid vs Not paid with a fico score of 699 and below:')
sb_f699 = []
for item in sb_payment_fico_699_grp['fico'].count():
    sb_f699.append(item)
sb_f699_sum = sum(sb_f699)
sb_perc_f699=[]
for item in sb_f699:
    b699=round(item/sb_f699_sum,2)
    print(b699)
    sb_perc_f699.append(b699)

sb_payment_fico_799_grp = sb_fico_799.groupby('not_fully_paid')
print('\nPercentage of Paid vs Not Paid with a fico score of 700-799:')
sb_f799 = []
for item in sb_payment_fico_799_grp['fico'].count():
    sb_f799.append(item)
sb_f799_sum = sum(sb_f799)
sb_perc_f799=[]
for item in sb_f799:
    b799=round(item/sb_f799_sum,2)
    print(b799)
    sb_perc_f799.append(b799)

sb_payment_fico_899_grp = sb_fico_899.groupby('not_fully_paid')
print('\nPercentage of Paid vs Not Paid with a fico score of 800 and above:')
sb_f899 = []
for item in sb_payment_fico_899_grp['fico'].count():
    sb_f899.append(item)
sb_f899_sum = sum(sb_f899)
sb_perc_f899=[]
for item in sb_f899:
    b899=round(item/sb_f899_sum,2)
    print(b899)
    sb_perc_f899.append(b899)
```

```
fico: The self-reported annual income of the borrower.
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

Percentage of Paid vs Not paid with a fico score of 699 and below:
0.62
0.38

Percentage of Paid vs Not Paid with a fico score of 700-799:
0.77
0.23

Percentage of Paid vs Not Paid with a fico score of 800 and above:
0.92
```

A graph was created to show the increase in loans paid back and the decrease in Defaulted loans

```
import matplotlib.pyplot as plt

names = ['699\n and below \npaid','699\n and below \ndefaulted',
         '700-799\npaid','700-799\ndefaulted',
         '800\n and above\npaid','800\n and above\ndefaulted']
values = [sb_perc_f699[0],sb_perc_f699[1],sb_perc_f799[0],sb_perc_f799[1],sb_perc_f899[0],
          sb_perc_f899[1]]
colors = ['green','red','green','red','green','red']
plt.figure(figsize=(25, 6))

plt.subplot(131)
plt.bar(names, values, color= colors)
plt.ylabel('Percent')
plt.xlabel('Fico Credit Score')
plt.title('Fico Credit Score and small business loans')

plt.show()
```

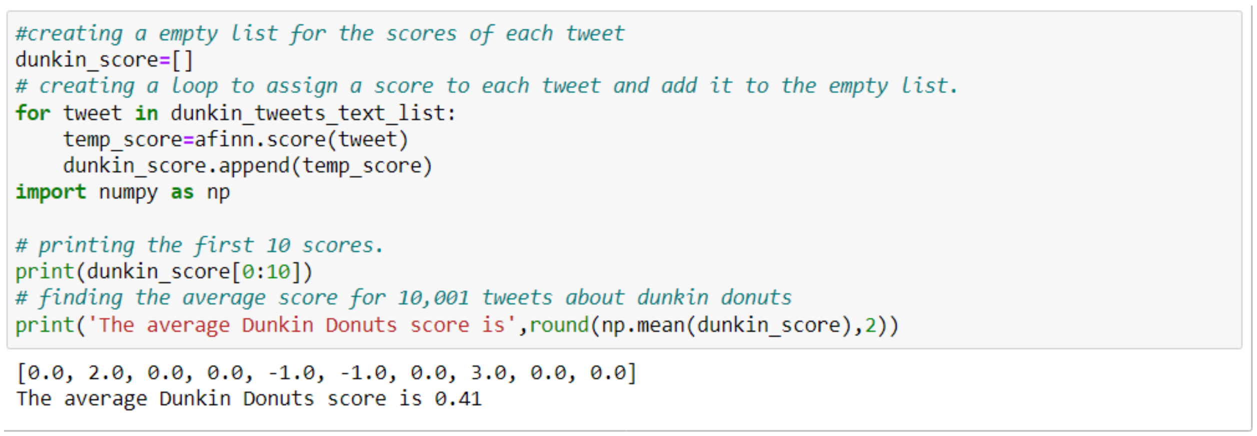


As the fico credit score increases from one bin to the next the percentage of defaulted loans decreases. Showing that fico credit score is an important indicator of how often a small business loan defaults. When comparing the two graphs the defaulting loans are much higher in the below 699 the 700-799 groups but remain the same in the above 800 grouping. Showing that having a higher fico credit score is a strong indicator for a potential small business loan recipient.

Sentiment Analysis (Tweets)

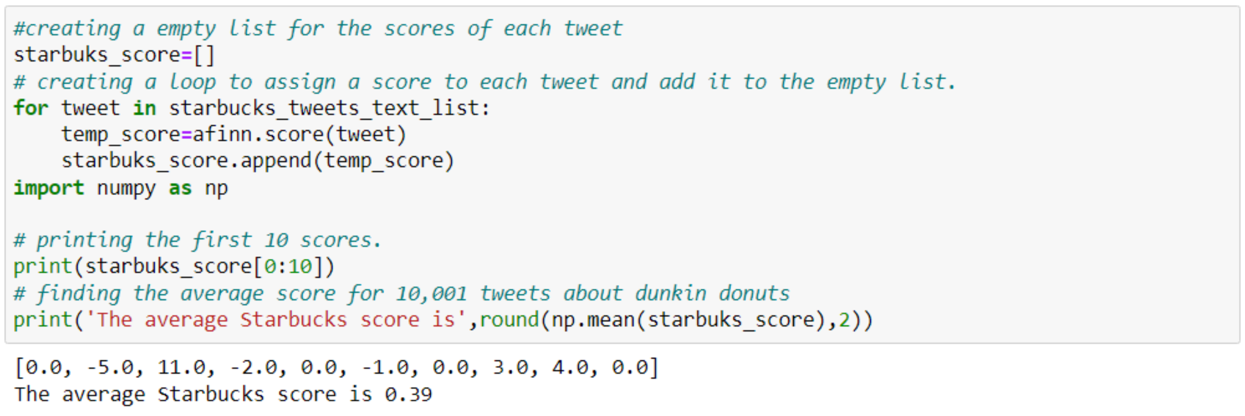
Question #6 -*Can sentiment analysis be used to increase knowledge and predict if a small business loan will default?

Using afinn this list was passed through a loop and a sentiment score was assigned to each tweet. This score determined if the tweet was a positive statement or a negative statement. An average of all the tweets was obtained.



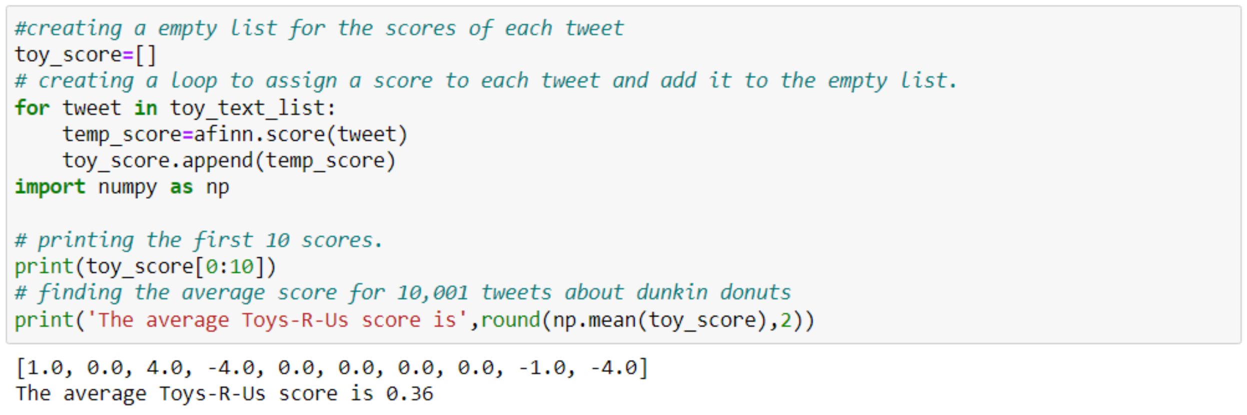
```
#creating a empty list for the scores of each tweet
dunkin_score=[]
# creating a loop to assign a score to each tweet and add it to the empty list.
for tweet in dunkin_tweets_text_list:
    temp_score=afinn.score(tweet)
    dunkin_score.append(temp_score)
import numpy as np

# printing the first 10 scores.
print(dunkin_score[0:10])
# finding the average score for 10,001 tweets about dunkin donuts
print('The average Dunkin Donuts score is',round(np.mean(dunkin_score),2))
```

```
[0.0, 2.0, 0.0, 0.0, -1.0, -1.0, 0.0, 3.0, 0.0, 0.0]
The average Dunkin Donuts score is 0.41
```



```
#creating a empty list for the scores of each tweet
starbuks_score=[]
# creating a loop to assign a score to each tweet and add it to the empty list.
for tweet in starbucks_tweets_text_list:
    temp_score=afinn.score(tweet)
    starbuks_score.append(temp_score)
import numpy as np

# printing the first 10 scores.
print(starbuks_score[0:10])
# finding the average score for 10,001 tweets about dunkin donuts
print('The average Starbucks score is',round(np.mean(starbuks_score),2))
```

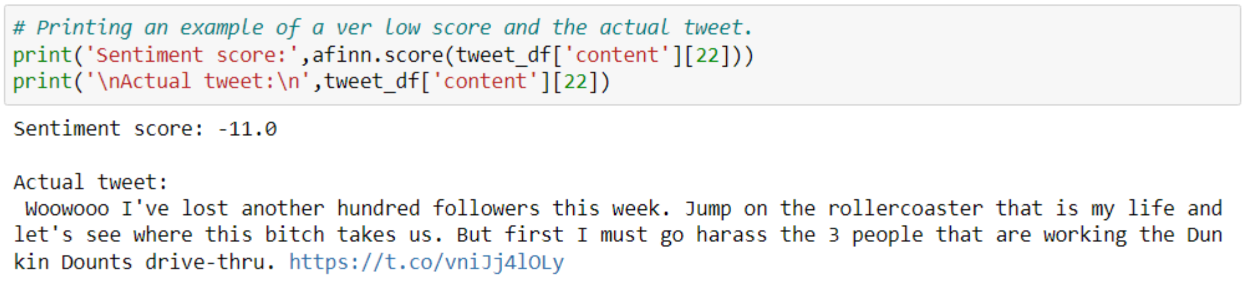
```
[0.0, -5.0, 11.0, -2.0, 0.0, -1.0, 0.0, 3.0, 4.0, 0.0]
The average Starbucks score is 0.39
```



```
#creating a empty list for the scores of each tweet
toy_score=[]
# creating a loop to assign a score to each tweet and add it to the empty list.
for tweet in toy_text_list:
    temp_score=afinn.score(tweet)
    toy_score.append(temp_score)
import numpy as np

# printing the first 10 scores.
print(toy_score[0:10])
# finding the average score for 10,001 tweets about dunkin donuts
print('The average Toys-R-Us score is',round(np.mean(toy_score),2))
```

```
[1.0, 0.0, 4.0, -4.0, 0.0, 0.0, 0.0, 0.0, -1.0, -4.0]
The average Toys-R-Us score is 0.36
```

The average scores are all very similar in value. To further examine this data a very negative Dunkin Donuts tweet was selected, and its contents were viewed manually.



```
# Printing an example of a ver low score and the actual tweet.
print('Sentiment score:',afinn.score(tweet_df['content'][22]))
print('\nActual tweet:\n',tweet_df['content'][22])
```

```
Sentiment score: -11.0

Actual tweet:
 Woowooo I've lost another hundred followers this week. Jump on the rollercoaster that is my life and
let's see where this bitch takes us. But first I must go harass the 3 people that are working the Dun
kin Dounts drive-thru. https://t.co/vniJj4lOLy
```

While this is a very negative tweet, it isn't implying dissatisfaction Dunkin Donuts. While the user is expressing negativity towards what is going on and how he is going to channel this towards the drive thru workers at Dunkin Donuts, it is not a negative reflection on the brand or business.

Since Sentiment analysis gives a score based on simply negative or positive words it is not what we were looking for to convey if a business is in financial trouble that could increase the risk of defaulting on a loan.

Regular Expression (Tweets)

Question #7 -*Can regular expressions be used to increase knowledge and predict if a small business loan will default?

Phrases were put together that could be used to indicate if a business would be at higher risk of defaulting on a loan. This was used with regular expressions the pull tweets that contained these phrases out of the list of tweets and append them to another list. A message and the number of tweets is displayed along with the tweets. Below is two business Starbucks followed by Dunkin Donuts.

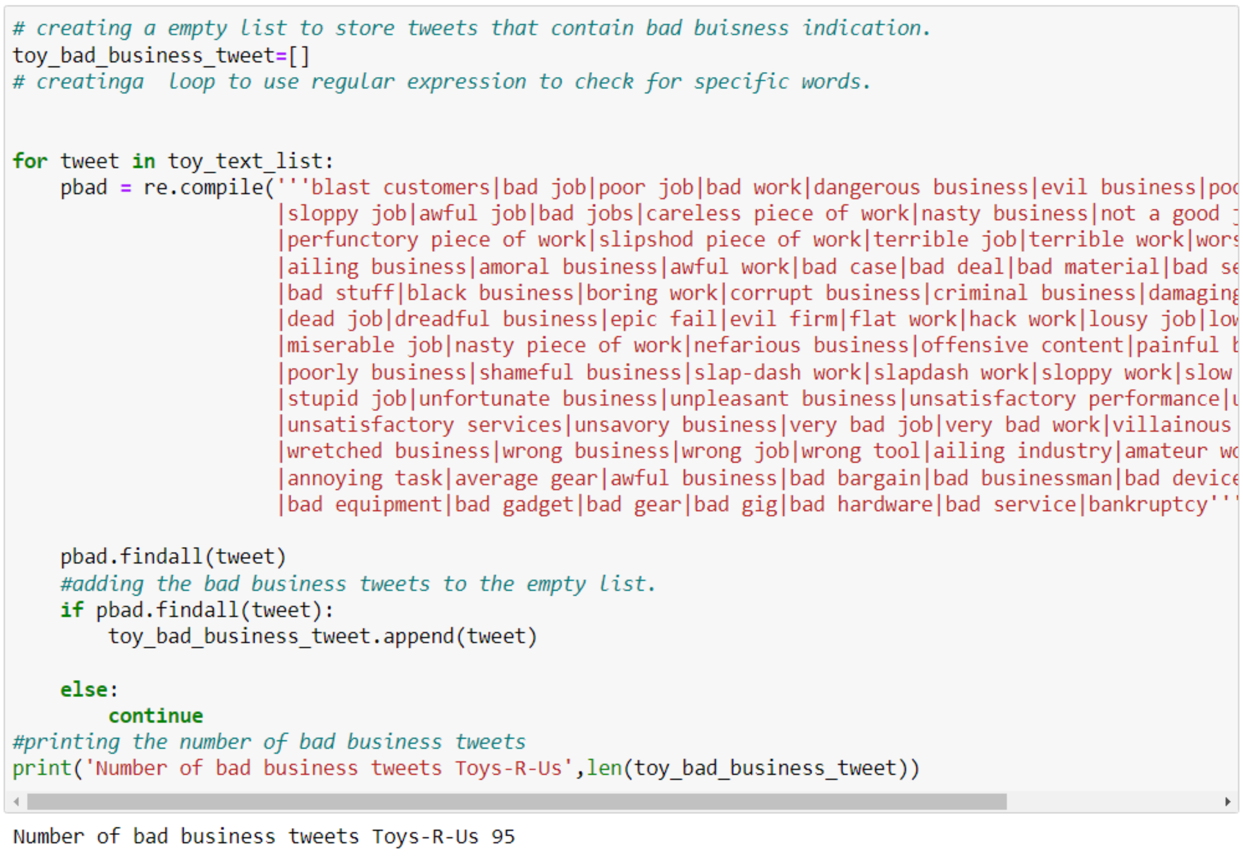
```python
# creating a empty list to store tweets that contain bad buisness indication.
sb_bad_business_tweet=[]
# creatinga  loop to use regular expression to check for specific words.
for tweet_2 in starbucks_tweets_text_list:
    pbad = re.compile('''blast customers|bad job|poor job|bad work|dangerous business|evil business|poo
                      |sloppy job|awful job|bad jobs|careless piece of work|nasty business|not a good j
                      |perfunctory piece of work|slipshod piece of work|terrible job|terrible work|wors
                      |ailing business|amoral business|awful work|bad case|bad deal|bad material|bad se
                      |bad stuff|black business|boring work|corrupt business|criminal business|damaging
                      |dead job|dreadful business|epic fail|evil firm|flat work|hack work|lousy job|low
                      |miserable job|nasty piece of work|nefarious business|offensive content|painful b
                      |poorly business|shameful business|slap-dash work|slapdash work|sloppy work|slow
                      |stupid job|unfortunate business|unpleasant business|unsatisfactory performance|u
                      |unsatisfactory services|unsavory business|very bad job|very bad work|villainous
                      |wretched business|wrong business|wrong job|wrong tool|ailing industry|amateur wo
                      |annoying task|average gear|awful business|bad bargain|bad businessman|bad device
                      |bad equipment|bad gadget|bad gear|bad gig|bad hardware''')

    pbad.findall(tweet_2)
    #adding the bad business tweets to the empty list.
    if pbad.findall(tweet_2):
        sb_bad_business_tweet.append(tweet_2)
    else:
        continue
#printing the number of bad business tweets
print('Number of bad business tweets about Starbucks',len(sb_bad_business_tweet))
# printing the first 10 bad business tweets
print(sb_bad_business_tweet[:10])
```

```
Number of bad business tweets about Starbucks 0
[]
```

```python
import re
# creating a empty list to store tweets that contain bad buisness indication.
dd_bad_business_tweet=[]
# creatinga  loop to use regular expression to check for specific words.
for tweet in dunkin_tweets_text_list:
    pbad = re.compile('''blast customers|bad job|poor job|bad work|dangerous business|evil business|poo
                      |sloppy job|awful job|bad jobs|careless piece of work|nasty business|not a good j
                      |perfunctory piece of work|slipshod piece of work|terrible job|terrible work|wors
                      |ailing business|amoral business|awful work|bad case|bad deal|bad material|bad se
                      |bad stuff|black business|boring work|corrupt business|criminal business|damaging
                      |dead job|dreadful business|epic fail|evil firm|flat work|hack work|lousy job|low
                      |miserable job|nasty piece of work|nefarious business|offensive content|painful b
                      |poorly business|shameful business|slap-dash work|slapdash work|sloppy work|slow
                      |stupid job|unfortunate business|unpleasant business|unsatisfactory performance|u
                      |unsatisfactory services|unsavory business|very bad job|very bad work|villainous
                      |wretched business|wrong business|wrong job|wrong tool|ailing industry|amateur wo
                      |annoying task|average gear|awful business|bad bargain|bad businessman|bad device
                      |bad equipment|bad gadget|bad gear|bad gig|bad hardware''')
    pbad.findall(tweet)
    #adding the bad business tweets to the empty list.
    if pbad.findall(tweet):
        dd_bad_business_tweet.append(tweet)
    else:
        continue
#printing the number of bad business tweets
print('Number of bad business tweets about Dunkin Donuts',len(dd_bad_business_tweet))
# printing the first 10 bad business tweets
print(dd_bad_business_tweet[:10])
```

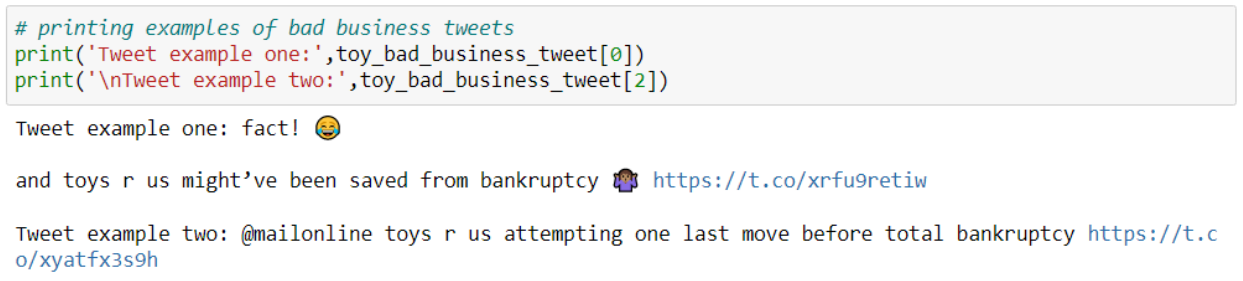
```
Number of bad business tweets 1
["“@phillypicks: revealed - twitter's top philadelphia cafes: http://t.co/rqztnznb” //epic fail #dunk
indounts is not a cafe."]
```

No tweets containing the indicated phrases were found in the Starbuck's tweets. There was one found in the Dunkin Donut's tweets. This was not actually an indication of higher risk of defaulting. Since it was just one tweet it can easily be viewed meaning that this process still can work as intended. Since Toys-R-Us is a bankrupt company, we would expect to find some tweets that actually have indications that this company is high risk for defaulting on loans.



```
# creating a empty list to store tweets that contain bad buisness indication.
toy_bad_business_tweet=[]
# creatinga  loop to use regular expression to check for specific words.


for tweet in toy_text_list:
    pbad = re.compile('''blast customers|bad job|poor job|bad work|dangerous business|evil business|poo
                       |sloppy job|awful job|bad jobs|careless piece of work|nasty business|not a good j
                       |perfunctory piece of work|slipshod piece of work|terrible job|terrible work|wors
                       |ailing business|amoral business|awful work|bad case|bad deal|bad material|bad se
                       |bad stuff|black business|boring work|corrupt business|criminal business|damaging
                       |dead job|dreadful business|epic fail|evil firm|flat work|hack work|lousy job|low
                       |miserable job|nasty piece of work|nefarious business|offensive content|painful b
                       |poorly business|shameful business|slap-dash work|slapdash work|sloppy work|slow
                       |stupid job|unfortunate business|unpleasant business|unsatisfactory performance|u
                       |unsatisfactory services|unsavory business|very bad job|very bad work|villainous
                       |wretched business|wrong business|wrong job|wrong tool|ailing industry|amateur wo
                       |annoying task|average gear|awful business|bad bargain|bad businessman|bad device
                       |bad equipment|bad gadget|bad gear|bad gig|bad hardware|bad service|bankruptcy'''

    pbad.findall(tweet)
    #adding the bad business tweets to the empty list.
    if pbad.findall(tweet):
        toy_bad_business_tweet.append(tweet)

    else:
        continue
#printing the number of bad business tweets
print('Number of bad business tweets Toys-R-Us',len(toy_bad_business_tweet))
```

```
Number of bad business tweets Toys-R-Us 95
```



```
# printing examples of bad business tweets
print('Tweet example one:',toy_bad_business_tweet[0])
print('\nTweet example two:',toy_bad_business_tweet[2])
```

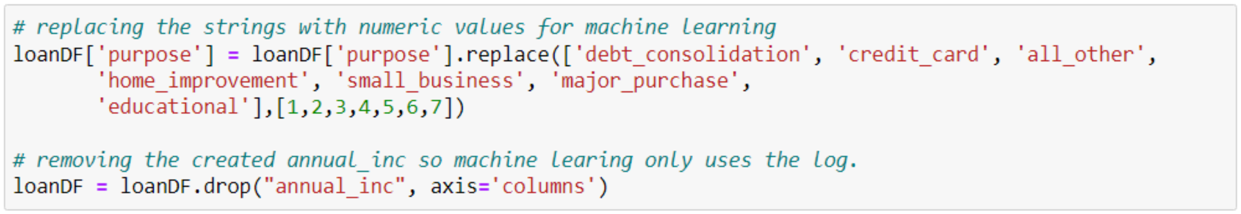
```
Tweet example one: fact! 😂

and toys r us might've been saved from bankruptcy 🤷‍♀️ https://t.co/xrfu9retiw

Tweet example two: @mailonline toys r us attempting one last move before total bankruptcy https://t.c
o/xyatfx3s9h
```

This produced actual tweets that indicate that Toys-R-Us (a bankrupt company) is a higher risk of defaulting on loans.

Machine Learning

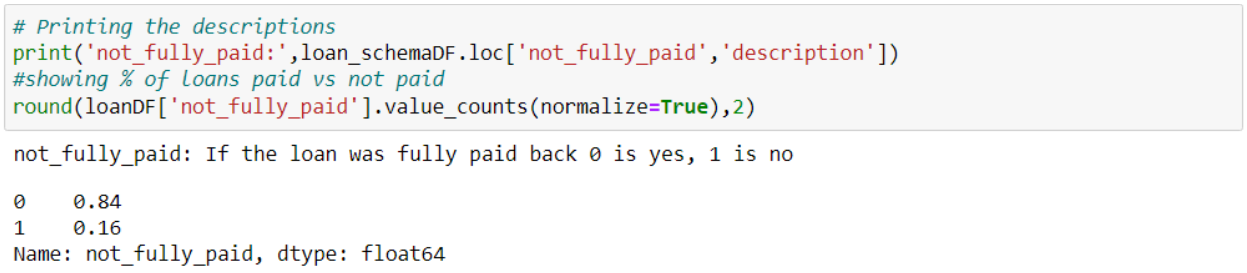
Question #8 -*Can machine learning provide a way to predict if an applicant will default on a loan?

To use machine learning a few cleaning issues need to be resolved. First the purpose column needed to be converted to numeric. Then the created annual_inc column needed to be removed so the algorithms will use only the log_annual_inc column for predicting.



```
# replacing the strings with numeric values for machine learning
loanDF['purpose'] = loanDF['purpose'].replace(['debt_consolidation', 'credit_card', 'all_other',
       'home_improvement', 'small_business', 'major_purchase',
       'educational'],[1,2,3,4,5,6,7])

# removing the created annual_inc so machine learing only uses the log.
loanDF = loanDF.drop("annual_inc", axis='columns')
```

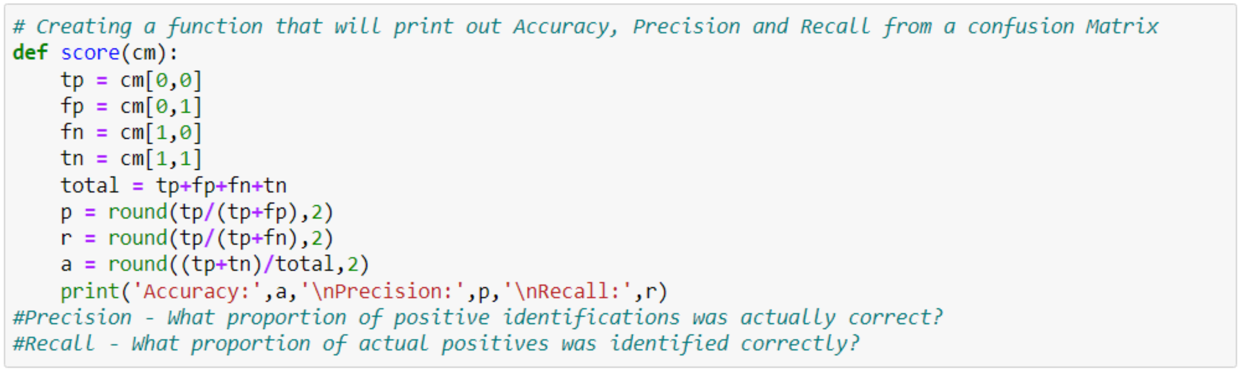
The actual percentage of paid vs defaulted loans that the lending club loan experts achieved was found in order to compare it to the predictive models accuracy.



```
# Printing the descriptions
print('not_fully_paid:',loan_schemaDF.loc['not_fully_paid','description'])
#showing % of loans paid vs not paid
round(loanDF['not_fully_paid'].value_counts(normalize=True),2)
```

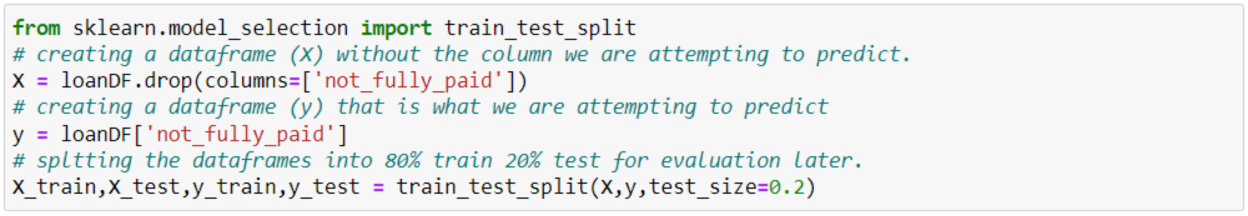
```
not_fully_paid: If the loan was fully paid back 0 is yes, 1 is no

0    0.84
1    0.16
Name: not_fully_paid, dtype: float64
```

A function called score() was created to find all the metrics of interest from the confusion matrix created by sklearn. These metrics are Accuracy, Precision and Recall. Accuracy is simply the amount predicted right. If the model predicted, the borrower would pay the loan back and they did, its right. If the model predicted the borrower would default and they did, its right. Another way to say this is true positive plus true negative divided by the total. Precision is looking at the number of times the predicted model was correct, did it give false positives? Did the model predict this person would pay a loan back, but they defaulted? Precision is True positive divided by the sum of true positive and false positive. The last metric is Recall which is basically how much of the positives did the model identify correctly. Did the model say a person would default when they paid their loan in full? This is true positive divided by the sum of true positive and false negative.



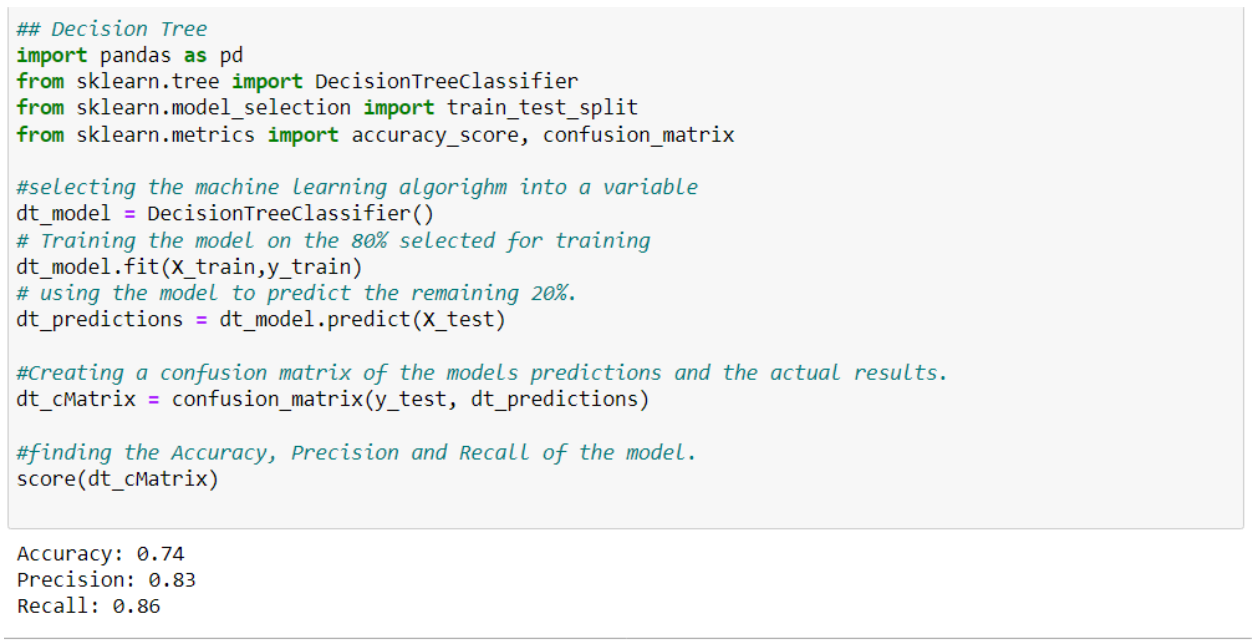
```
# Creating a function that will print out Accuracy, Precision and Recall from a confusion Matrix
def score(cm):
    tp = cm[0,0]
    fp = cm[0,1]
    fn = cm[1,0]
    tn = cm[1,1]
    total = tp+fp+fn+tn
    p = round(tp/(tp+fp),2)
    r = round(tp/(tp+fn),2)
    a = round((tp+tn)/total,2)
    print('Accuracy:',a,'\nPrecision:',p,'\nRecall:',r)
#Precision - What proportion of positive identifications was actually correct?
#Recall - What proportion of actual positives was identified correctly?
```

The data frame needed to be broken up into predictors and what was being predicted. Then split into test and training sets. First it was broken into X being the predictors (not_fully_paid column dropped) and y being what is being predicted (not_fully_paid column). Using sklearn train_test_split both these data frames were split 80% for training the models and 20% for testing the models and comparison.



```
from sklearn.model_selection import train_test_split
# creating a dataframe (X) without the column we are attempting to predict.
X = loanDF.drop(columns=['not_fully_paid'])
# creating a dataframe (y) that is what we are attempting to predict
y = loanDF['not_fully_paid']
# spltting the dataframes into 80% train 20% test for evaluation later.
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

Decision Tree was used first as an easy start with modeling. The predictions were placed in a confusion matrix and that was run through the score() function created.

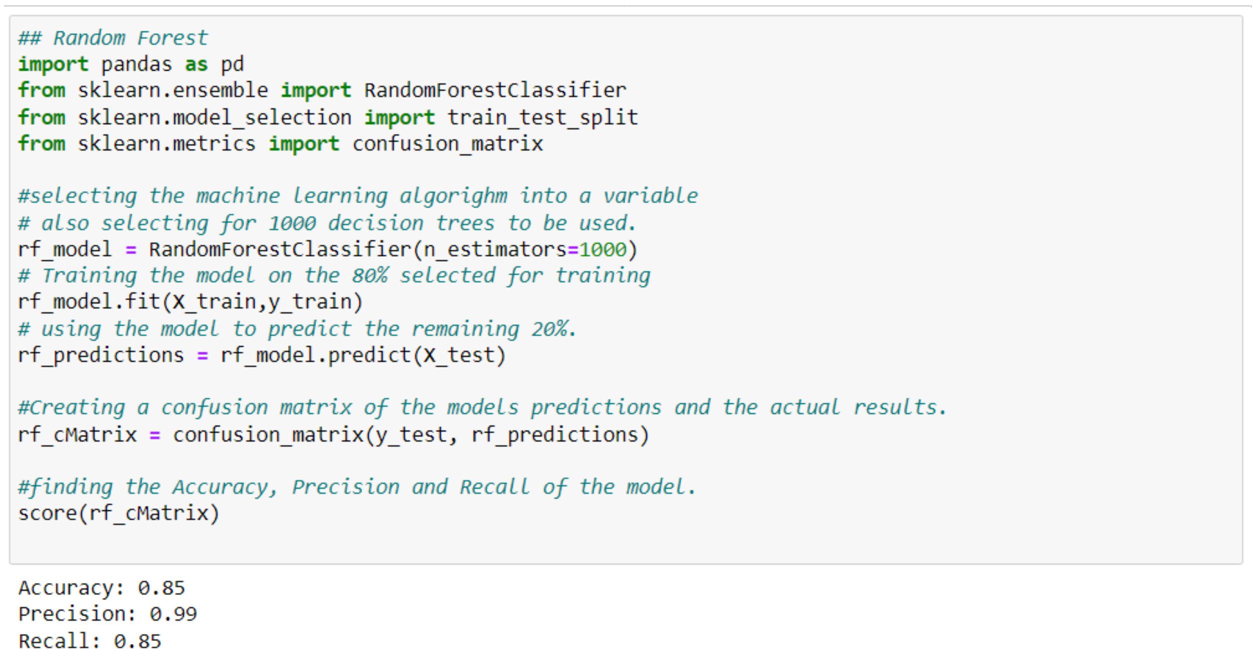


```python
## Decision Tree
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

#selecting the machine learning algorighm into a variable
dt_model = DecisionTreeClassifier()
# Training the model on the 80% selected for training
dt_model.fit(X_train,y_train)
# using the model to predict the remaining 20%.
dt_predictions = dt_model.predict(X_test)

#Creating a confusion matrix of the models predictions and the actual results.
dt_cMatrix = confusion_matrix(y_test, dt_predictions)

#finding the Accuracy, Precision and Recall of the model.
score(dt_cMatrix)
```

```
Accuracy: 0.74
Precision: 0.83
Recall: 0.86
```

This model was found not to be as accurate as the banking experts at lending club.

Random Forest was used and tuned to use 1000 different Decision Trees.

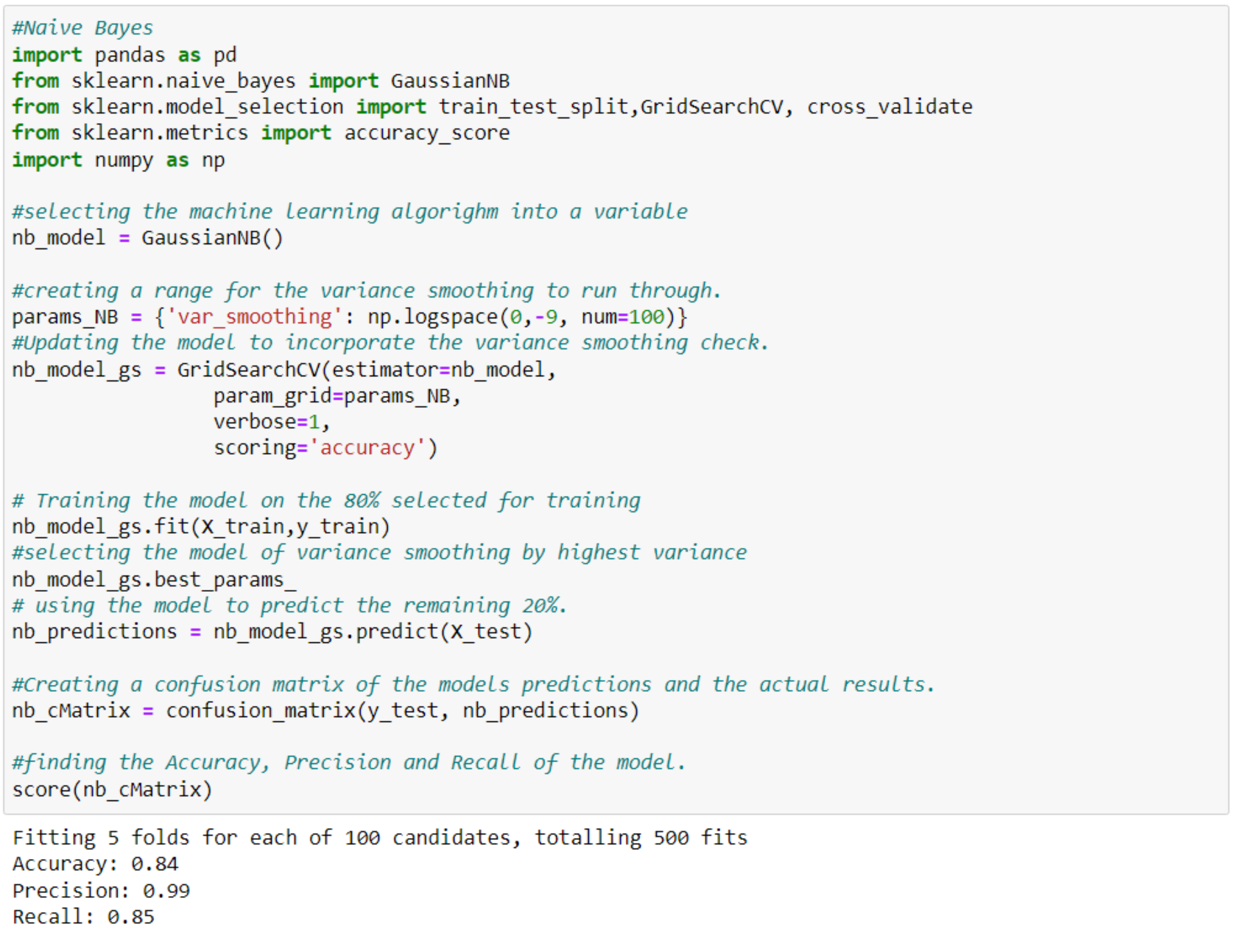


```python
## Random Forest
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

#selecting the machine learning algorighm into a variable
# also selecting for 1000 decision trees to be used.
rf_model = RandomForestClassifier(n_estimators=1000)
# Training the model on the 80% selected for training
rf_model.fit(X_train,y_train)
# using the model to predict the remaining 20%.
rf_predictions = rf_model.predict(X_test)

#Creating a confusion matrix of the models predictions and the actual results.
rf_cMatrix = confusion_matrix(y_test, rf_predictions)

#finding the Accuracy, Precision and Recall of the model.
score(rf_cMatrix)
```

```
Accuracy: 0.85
Precision: 0.99
Recall: 0.85
```

This was found to be slightly more accurate than the lending club experts on the testing data and had almost no false positives.

Naïve Bayes was used and tuned using a range of variance smoothing. This allows for multiple models to be created and the one with the highest accuracy is chosen and the rest discarded.

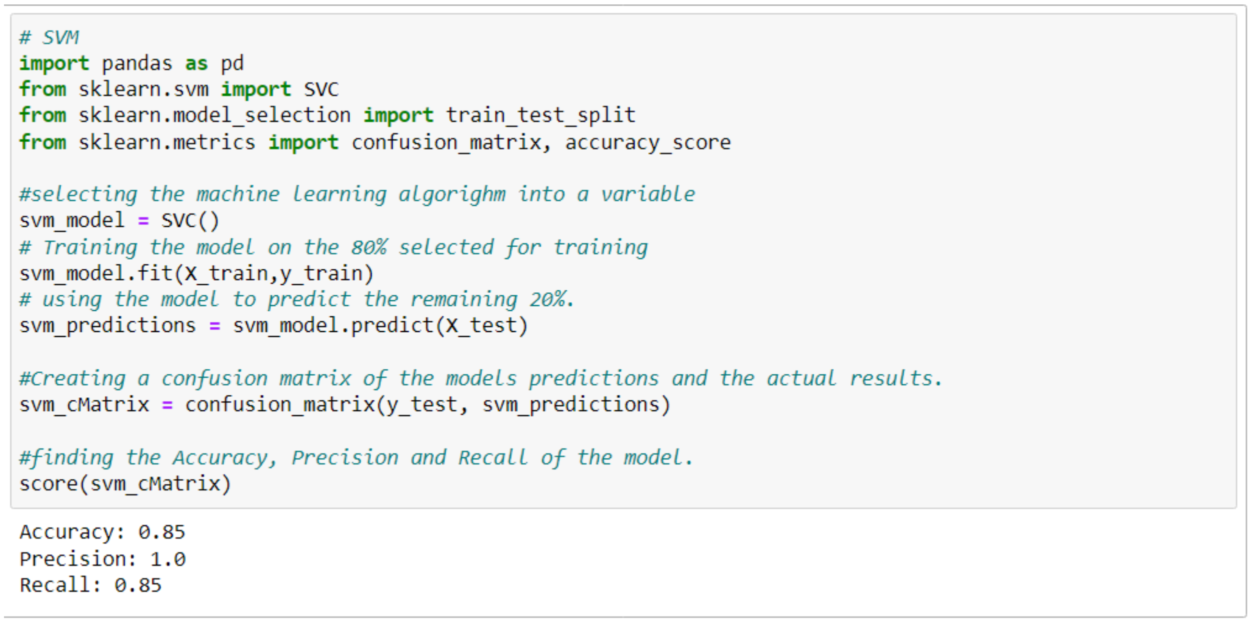


```python
#Naive Bayes
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split,GridSearchCV, cross_validate
from sklearn.metrics import accuracy_score
import numpy as np

#selecting the machine learning algorighm into a variable
nb_model = GaussianNB()

#creating a range for the variance smoothing to run through.
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
#Updating the model to incorporate the variance smoothing check.
nb_model_gs = GridSearchCV(estimator=nb_model,
                 param_grid=params_NB,
                 verbose=1,
                 scoring='accuracy')

# Training the model on the 80% selected for training
nb_model_gs.fit(X_train,y_train)
#selecting the model of variance smoothing by highest variance
nb_model_gs.best_params_
# using the model to predict the remaining 20%.
nb_predictions = nb_model_gs.predict(X_test)

#Creating a confusion matrix of the models predictions and the actual results.
nb_cMatrix = confusion_matrix(y_test, nb_predictions)

#finding the Accuracy, Precision and Recall of the model.
score(nb_cMatrix)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
Accuracy: 0.84
Precision: 0.99
Recall: 0.85
```

The Naïve Bayes was found to be slightly less accurate than the Random Forest and had similar precision.

The last predictive model created was a Support Vector Machine.

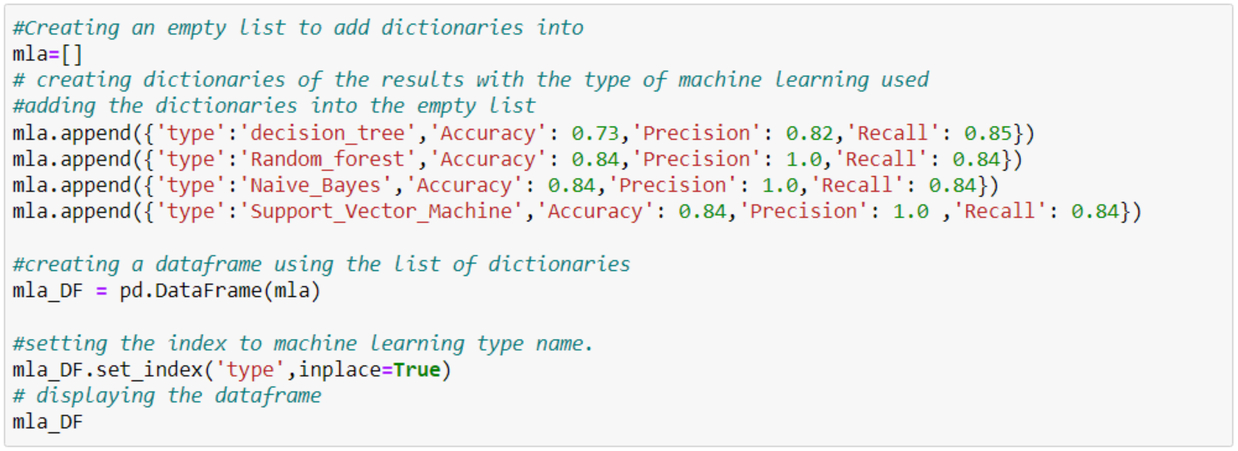


```
# SVM
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

#selecting the machine learning algorighm into a variable
svm_model = SVC()
# Training the model on the 80% selected for training
svm_model.fit(X_train,y_train)
# using the model to predict the remaining 20%.
svm_predictions = svm_model.predict(X_test)

#Creating a confusion matrix of the models predictions and the actual results.
svm_cMatrix = confusion_matrix(y_test, svm_predictions)

#finding the Accuracy, Precision and Recall of the model.
score(svm_cMatrix)
```

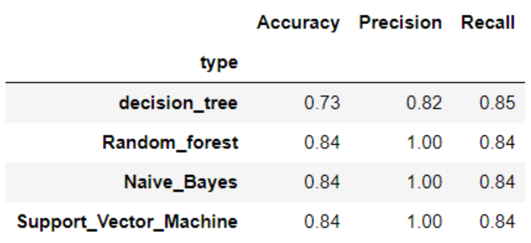
```
Accuracy: 0.85
Precision: 1.0
Recall: 0.85
```

The SVM was found to have the same accuracy as the Random Forest but with zero false positives.

A list was created, and a dictionary of each model was inserted using type, Accuracy, Precision, and Recall as the keys, and the name and results as the values. This list was then converted into a pandas data frame.

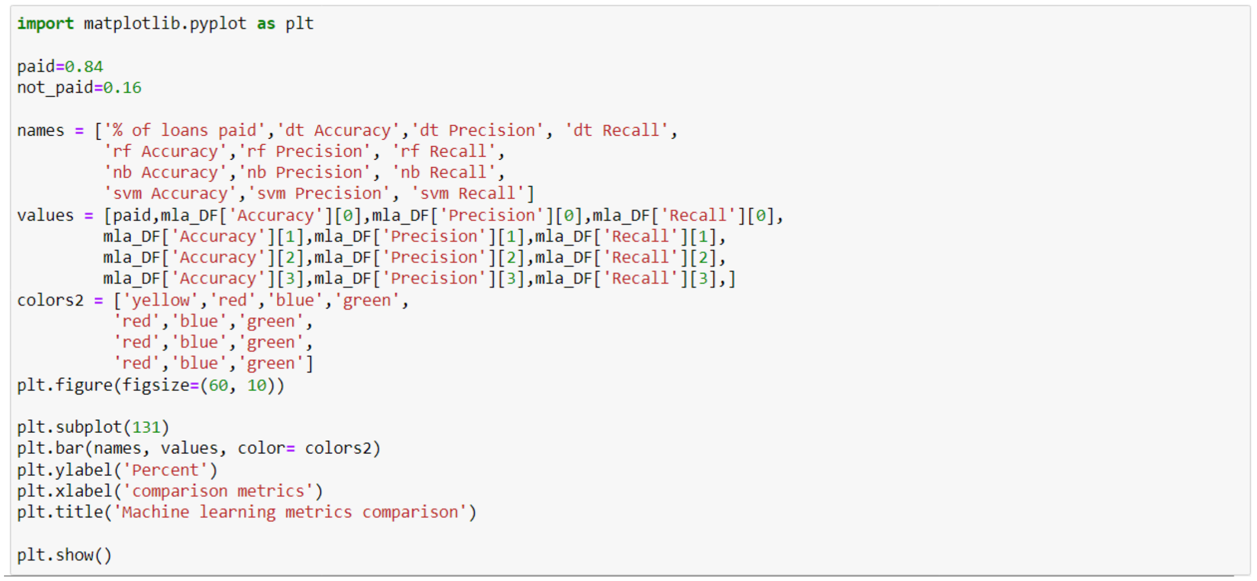


```
#Creating an empty list to add dictionaries into
mla=[]
# creating dictionaries of the results with the type of machine learning used
#adding the dictionaries into the empty list
mla.append({'type':'decision_tree','Accuracy': 0.73,'Precision': 0.82,'Recall': 0.85})
mla.append({'type':'Random_forest','Accuracy': 0.84,'Precision': 1.0,'Recall': 0.84})
mla.append({'type':'Naive_Bayes','Accuracy': 0.84,'Precision': 1.0,'Recall': 0.84})
mla.append({'type':'Support_Vector_Machine','Accuracy': 0.84,'Precision': 1.0 ,'Recall': 0.84})

#creating a dataframe using the list of dictionaries
mla_DF = pd.DataFrame(mla)

#setting the index to machine learning type name.
mla_DF.set_index('type',inplace=True)
# displaying the dataframe
mla_DF
```

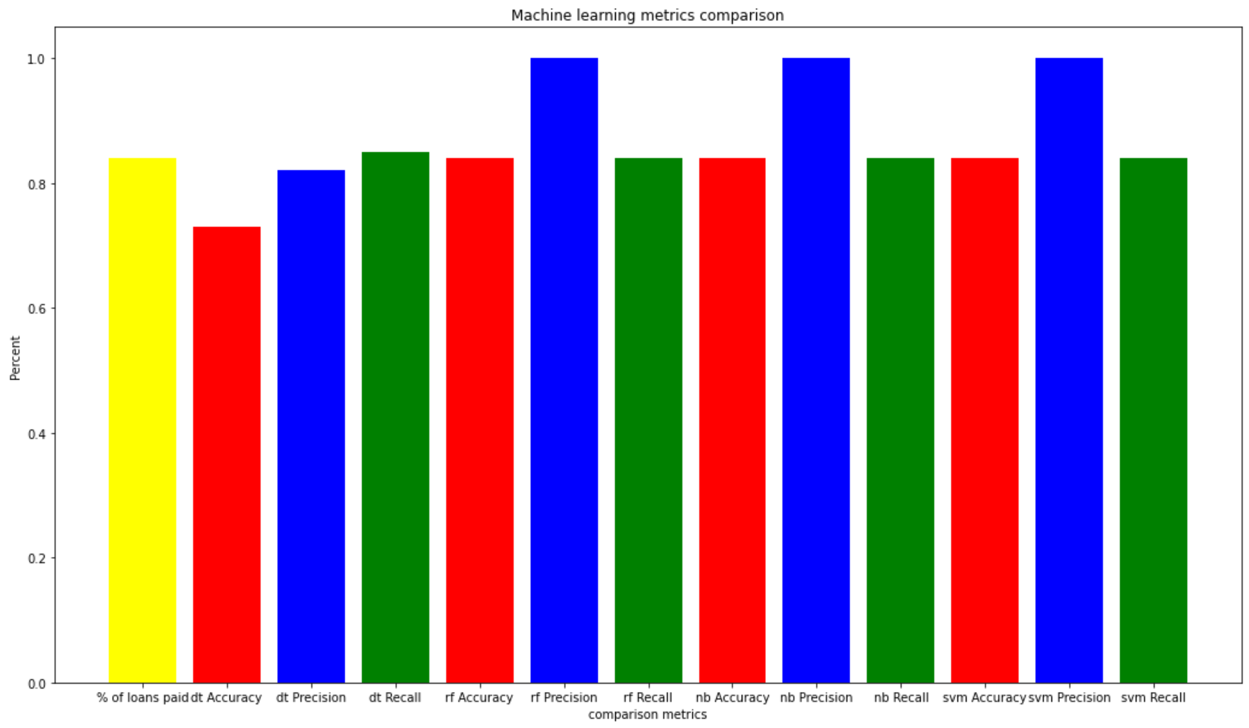


	Accuracy	Precision	Recall
type			
decision_tree	0.73	0.82	0.85
Random_forest	0.84	1.00	0.84
Naive_Bayes	0.84	1.00	0.84
Support_Vector_Machine	0.84	1.00	0.84

This data frame was graphed along with the original accuracy of the lending club experts.



```
import matplotlib.pyplot as plt

paid=0.84
not_paid=0.16

names = ['% of loans paid','dt Accuracy','dt Precision', 'dt Recall',
         'rf Accuracy','rf Precision', 'rf Recall',
         'nb Accuracy','nb Precision', 'nb Recall',
         'svm Accuracy','svm Precision', 'svm Recall']
values = [paid,mla_DF['Accuracy'][0],mla_DF['Precision'][0],mla_DF['Recall'][0],
         mla_DF['Accuracy'][1],mla_DF['Precision'][1],mla_DF['Recall'][1],
         mla_DF['Accuracy'][2],mla_DF['Precision'][2],mla_DF['Recall'][2],
         mla_DF['Accuracy'][3],mla_DF['Precision'][3],mla_DF['Recall'][3],]
colors2 = ['yellow','red','blue','green',
          'red','blue','green',
          'red','blue','green',
          'red','blue','green']
plt.figure(figsize=(60, 10))

plt.subplot(131)
plt.bar(names, values, color= colors2)
plt.ylabel('Percent')
plt.xlabel('comparison metrics')
plt.title('Machine learning metrics comparison')

plt.show()
```



This graph shows that machine learning can be a great tool to predict if a potential borrower will default on a loan or not.

Conclusion and next steps

We learned the following information:

1. How does the average monthly amount due change in loans that were paid back vs loans that were not paid?
   - The monthly amount due is higher for defaulted loans than loans paid in full. The amount is about $33. Yet we see that the mean is much higher than the median meaning there is outliers on the large side for both.
2. Does the purpose of the loan change the rate of being paid back or not?
   - Small business loans have a higher percentage of defaults compared to other loan types.
3. A. Does annual income change the percentage of loans being paid back?
   - As annual income increases the rate of defaulting decreases. Except when looking at excessing of $200,000 income which can be contributed to small business loans to those individuals.

   B. *What annual income ranges produces the least amount of defaulted small business loans?
   - Small business loans for income under $200,000 remains at smaller similar levels only increasing dramatically over the $200,000 income amount.
4. A. Does more debt-to-income ratio change the percentage of loans being paid back?
   - As the debt-to-income increases so does the chance the loan will be defaulted

   B. * How does debt-to-income ratio change with small business loans?
   - It simply become more severe. The number of defaults is more and increase with debt-to-income at a higher rate.
5. A. How does fico credit score change the percentage of loans being paid vs not?
   - As the fico credit score increases the percentage of loans defaulted drops.

   B. *What fico credit rating range produces the least amount of defaulted small business loans?
   - As the fico credit score increases the percentage of loans defaulting drops. The first two groupings (699 and below and 700-799) have a much higher defaulting rate in small business loans compared to all loans. Best range is above 800.

6. *Can sentiment analysis be used to increase knowledge and predict if a small business loan will default?
    - While we did get an overall score for the brands, the scores were not a real representation of the brands themselves. A more specific sentiment analysis looking for specific words is needed to get a better sentiment score that represents what we are looking for.
7. *Can regular expressions be used to increase knowledge and predict if a small business loan will default?
    - Yes, using regular expressions allows for tweets of interest (indicating higher risk of defaulting) to be removed and displayed.
8. *Can machine learning provide a way to predict if an applicant will default on a loan?
    - Machine learning can be a powerful tool to help predict if a person will default on a loan or not.
9. *Was anything found that significantly increase the chance that a small business loan does not default?
    - Yes, Fico credit score, Debt-to-income ratio and regular expressions being used on tweets about the business in question can help identify if a small business is a greater risk of defaulting. Also, the income of the potential borrower can indicate that a closer look at them may be required.

As for next steps more words and phrases would need to be added to the regular expressions work. There would also eventually be a system of scoring added to the words and phrases found to give the business a score for defaulting risk. The machine learning would require additional tuning by removing each column and rerunning the model. Then every possible combination of columns together and then every possible combination of 3, 4, etc... The best preforming model would then be used with the data obtained since 2015 to see if it was as accurate. If it held true, my suggestion would be to use the model to give loans and have the experts look at the loans it predicts as defaulting to find the false negatives.