In [1]:
```
'''Here we are studying customer review dateset.
we are using Zip file which is being taken from amazon website
and the name of the dateset is amazon reviews us Gift Card v1 00.The
 review dateset comprises of many customers'''
```

Out[1]:
```
'Here we are studying customer review dateset.\nwe are using Zip file
which is being taken from amazon website \nand the name of the dateset
is amazon reviews us Gift Card v1 00.The review dateset comprises of m
any customers'
```

In [2]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
import csv
import gzip
import matplotlib.pyplot as plt
from matplotlib import colors
from collections import defaultdict
f=gzip.open('amazon_reviews_us_Gift_Card_v1_00.tsv.gz','rt')
reader=csv.reader(f,delimiter='\t');header=next(reader)
dataset=[]
for line in reader:
  d=dict(zip(header,line))
  for field in['helpful_votes','star_rating','total_votes']:
    d[field]=int(d[field])
  for field in['verified_purchase','vine']:
    if d[field]=='Y':
      d[field]=True
    else:
      d[field]=False
  dataset.append(d)
```

In [3]:
```
'''each customer having his own review id
customer credentials details present such as which country
marketplace customer belongs to then various product details present
such as product id, product_category, product_parent, product_title
then details on customer reviews present such as review_body,
review_date,review_headline,review_id,customer star ratings,his total
votes,
his votes whether he found the store peoples helpful or not.
then details on whether purchase verified or not verified also given.
we now need to do some cleaning on the data so that our data becomes
 more meaningful
we should remove unnecessary and trivial data which
would only create more confusions and won't be helpful in
building significant data models and projecting accurate prediction
s'''
```

Out[3]:
```
"each customer having his own review id\ncustomer credentials details
present such as which country \nmarketplace customer belongs to then v
arious product details present \nsuch as product id, product_category,
product_parent, product_title\nthen details on customer reviews presen
t such as review_body, \nreview_date,review_headline,review_id,custome
r star ratings,his total votes,\nhis votes whether he found the store
peoples helpful or not.\nthen details on whether purchase verified or
not verified also given.\nwe now need to do some cleaning on the data
so that our data becomes more meaningful\nwe should remove unnecessary
and trivial data which \nwould only create more confusions and won't b
e helpful in \nbuilding significant data models and projecting accurat
e predictions"
```

In [4]:
```
dataset[0]
len(dataset)
```

Out[4]:
```
{'customer_id': '24371595',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False}
```

Out[4]:
```
148310
```

In [5]:
```
'''HERE WE TRY TO FILTER REVIEWS BY DATE.FOR THE MOMENT WE WILL FILTE
R BASED ON REVIEW'S YEAR
WE GOT ERROR.SO FIRST WE HAVE TO PREPROCESS OUR DATESET TO EXTRACT ON
LY THOSE
ENTRIES CONTAINING A REVIEW DATE FIELD'''
```

Out[5]:
```
"HERE WE TRY TO FILTER REVIEWS BY DATE.FOR THE MOMENT WE WILL FILTER B
ASED ON REVIEW'S YEAR\nWE GOT ERROR.SO FIRST WE HAVE TO PREPROCESS OUR
DATESET TO EXTRACT ONLY THOSE\nENTRIES CONTAINING A REVIEW DATE FIELD"
```

In [6]:
```
for d in dataset:
  d['yearint']=int(d['review_date'][:4])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-6-e9bbad8b7105> in <module>()
      1 for d in dataset:
----> 2   d['yearint']=int(d['review_date'][:4])

KeyError: 'review_date'
```

In [7]:
```
dataset=[d for d in dataset if 'review_date' in d ]
print('\n')
len(dataset)
```

Out[7]: 148309

In [8]:
```
#now let us filter old reviews i.e. those before 2010
```

In [9]:
```
for d in dataset:
  d['yearint']=int(d['review_date'][:4])
dataset=[d for d in dataset if d['yearint'] >2010]
dataset[0]
len(dataset)
```

Out[9]:
```
{'customer_id': '24371595',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[9]: 146727

In [10]:
```
#let us write other list comprehension to exclude reviews with low helpful rates
```

In [11]:
```
dataset=[d for d in dataset if d['total_votes']<3
         or d['helpful_votes']/d['total_votes']>=0.5]
dataset[0]
len(dataset)
```

Out[11]:
```
{'customer_id': '24371595',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[11]:
```
146461
```

In [12]:
```
'''let us filter our dataset to discard inactive users i.e.
users who have written only a single review in this directory.
then we can filter to keep users with 2 or more reviews'''
```

Out[12]:
```
'let us filter our dataset to discard inactive users i.e. \nusers who
have written only a single review in this directory.\nthen we can filt
er to keep users with 2 or more reviews'
```

In [13]:
```
nReviewperuser=defaultdict(int)
for d in dataset:
  nReviewperuser[d['customer_id']]+=1
dataset[0]
dataset=[d for d in dataset if nReviewperuser[d['customer_id']]>=2 ]
dataset[0]
len(dataset)
```

Out[13]:
```
{'customer_id': '24371595',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLIL5A',
 'product_parent': '346014806',
 'product_title': 'Amazon eGift Card - Celebrate',
 'review_body': 'Great birthday gift for a young adult.',
 'review_date': '2015-08-31',
 'review_headline': 'Five Stars',
 'review_id': 'R27ZP1F1CD0C3Y',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[13]:
```
{'customer_id': '48872127',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'BT00CTOYC0',
 'product_parent': '506740729',
 'product_title': 'Amazon.com $15 Gift Card in a Greeting Card (Amazon
Surprise Box Design)',
 'review_body': 'I love that I have instant, helpful options when I fo
rget a birthday! Thanks for saving the day Amazon!',
 'review_date': '2015-08-31',
 'review_headline': 'Quick Solution for Forgotten Occasion',
 'review_id': 'RVN4P3RU4F8IE',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[13]:
```
11048
```

In [14]:
```
#let us remove short reviews which may be uninformative
```

In [15]:
```
dataset=[d for d in dataset if len(d['review_body'].split())>=10]
dataset[0]
len(dataset)
```

Out[15]:
```
{'customer_id': '48872127',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'BT00CTOYC0',
 'product_parent': '506740729',
 'product_title': 'Amazon.com $15 Gift Card in a Greeting Card (Amazon
Surprise Box Design)',
 'review_body': 'I love that I have instant, helpful options when I fo
rget a birthday! Thanks for saving the day Amazon!',
 'review_date': '2015-08-31',
 'review_headline': 'Quick Solution for Forgotten Occasion',
 'review_id': 'RVN4P3RU4F8IE',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[15]:
```
6915
```

In [16]:
```
#average star rating for the entire dataset comes out as 4.806
```

In [17]:
```
ratings=[d['star_rating'] for d in dataset]
sum(ratings)/len(ratings)
```

Out[17]:
```
4.806073752711497
```

In [18]:
```
'''from the scatter plot we understand that helpful_votes
is positively correlated with total_votes i.e.
with increase in helpful votes total votes increase accordingly.'''
```

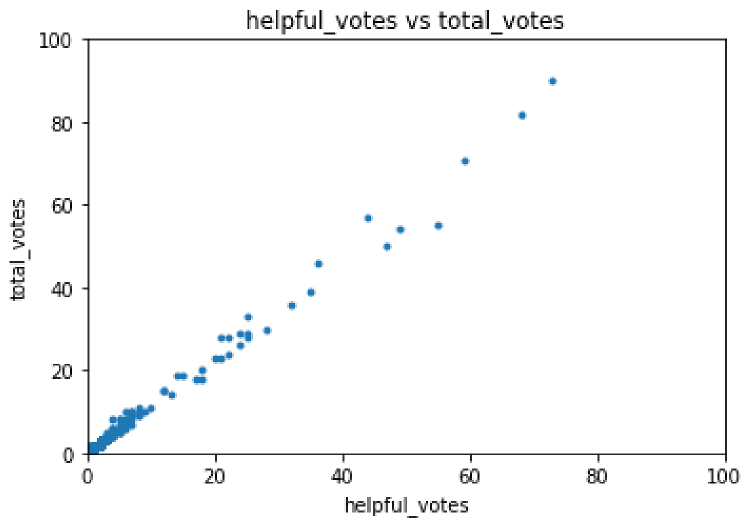
Out[18]:
```
'from the scatter plot we understand that helpful_votes \nis positivel
y correlated with total_votes i.e. \nwith increase in helpful votes to
tal votes increase accordingly.'
```

In [19]:
```
helpful_votes=[d['helpful_votes'] for d in dataset]
total_votes=[d['total_votes'] for d in dataset]
plt.gca().set(xlabel='helpful_votes',ylabel='total_votes',title='help
ful_votes vs total_votes')
plt.axis([0, 100, 0, 100])
size = 500
plt.scatter(helpful_votes,total_votes,marker=".")
```

Out[19]:
```
[Text(0, 0.5, 'total_votes'),
 Text(0.5, 0, 'helpful_votes'),
 Text(0.5, 1.0, 'helpful_votes vs total_votes')]
```

Out[19]:
```
(0.0, 100.0, 0.0, 100.0)
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x7f015677a4e0>
```



In [20]:
```
'''defaultdict"structure from the "collections"library allows us to a
utomate
initializing a dictionary with all zero counts'''
```

Out[20]:
```
'defaultdict"structure from the "collections"library allows us to auto
mate\ninitializing a dictionary with all zero counts'
```

In [21]:
```
#ratingcounts={1:0,2:0,3:0,4:0,5:0}
ratingcounts=defaultdict(int)
print('\n')
for d in dataset:
  ratingcounts[d['star_rating']]+=1
ratingcounts
```

Out[21]:
```
defaultdict(int, {1: 121, 2: 61, 3: 140, 4: 394, 5: 6199})
```

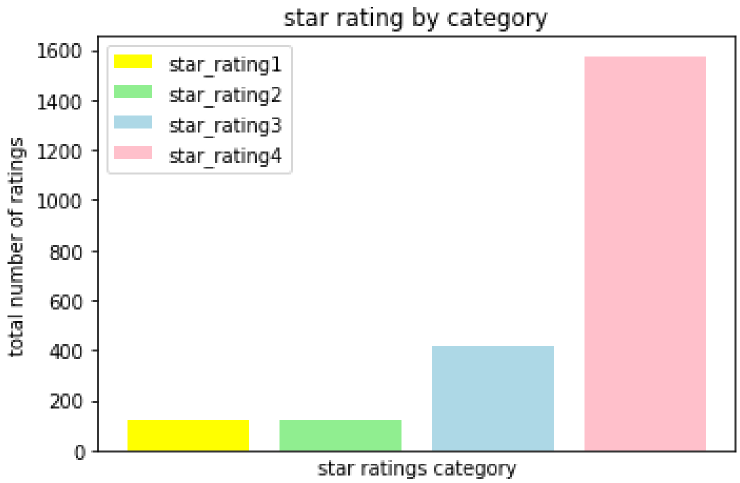
In [22]:

```
star1=sum([d['star_rating']for d in dataset if d['star_rating'] is 1
])
star2=sum([d['star_rating']for d in dataset if d['star_rating'] is 2
])
star3=sum([d['star_rating']for d in dataset if d['star_rating'] is 3
])
star4=sum([d['star_rating']for d in dataset if d['star_rating'] is 4
])
star5=sum([d['star_rating']for d in dataset if d['star_rating'] is 5
])
index=[1]
p1=plt.bar(index,star1,color='yellow')
index=[2]
p2=plt.bar(index,star2,color='lightgreen')
index=[3]
p3=plt.bar(index,star3,color='lightblue')
index=[4]
p4=plt.bar(index,star4,color='pink')
plt.gca().set(title='star rating by category',ylabel='total number of
ratings',xlabel='star ratings category')
plt.xticks([])
plt.legend((p1[0],p2[0],p3[0],p4[0]),('star_rating1','star_rating2',
'star_rating3','star_rating4'))
plt.show()
index=[1]
p5=plt.bar(index,star5,color='lightgreen')
index=[2]
p4=plt.bar(index,star4,color='pink')
plt.gca().set(title='star rating by category',ylabel='total number of
ratings',xlabel='star ratings category')
plt.xticks([])
plt.legend((p5[0],p4[0]),('star_rating5','star_rating4'))
plt.show()
```

```
Out[22]: [Text(0, 0.5, 'total number of ratings'),
          Text(0.5, 0, 'star ratings category'),
          Text(0.5, 1.0, 'star rating by category')]
```

```
Out[22]: ([], <a list of 0 Text major ticklabel objects>)
```

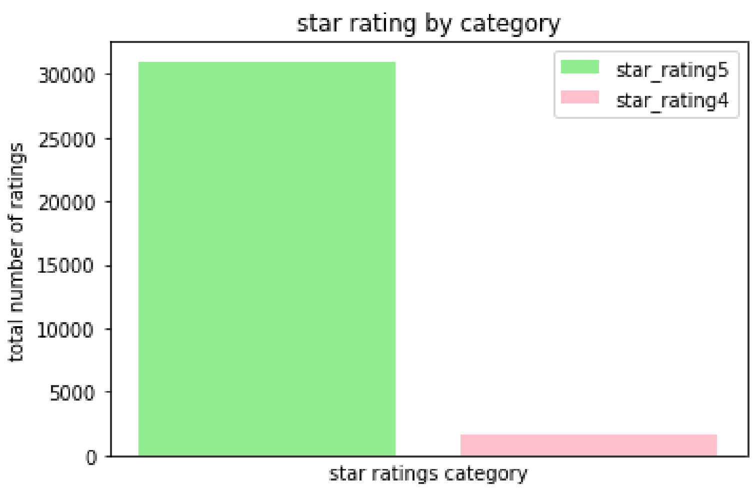
```
Out[22]: <matplotlib.legend.Legend at 0x7f01566b60f0>
```




```
Out[22]: [Text(0, 0.5, 'total number of ratings'),
          Text(0.5, 0, 'star ratings category'),
          Text(0.5, 1.0, 'star rating by category')]
```

```
Out[22]: ([], <a list of 0 Text major ticklabel objects>)
```

```
Out[22]: <matplotlib.legend.Legend at 0x7f01566c3320>
```




```
In [23]: '''from Bar plot we can clearly find that star_rating5 count is signi
         fiantly
         high comapared to other star_ratings'''
```

```
Out[23]: 'from Bar plot we can clearly find that star_rating5 count is signifia
         ntly\nhigh comapared to other star_ratings'
```

In [24]:
```
ratingsperproduct=defaultdict(list)
for d in dataset:
  ratingsperproduct[d['product_id']].append(d['star_rating'])
ratingsperproduct['B004LLIL5A'][-15:]
```

Out[24]:
```
[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]
```

In [25]:
```
#Average ratings per product stands out to be 4.8787
```

In [26]:
```
averageratingperproduct={}
for p in ratingsperproduct:
  averageratingperproduct[p]=sum(ratingsperproduct[p])/len(ratingsper
product[p])
averageratingperproduct['B004LLIL5A']
```

Out[26]:
```
4.878787878787879
```

In [27]:
```
toprated=[(averageratingperproduct[p],p) for p in averageratingperpro
duct
          if len(ratingsperproduct)>50]
toprated.sort()
toprated[201:211]
```

Out[27]:
```
[(4.785714285714286, 'B00CHQ7ESQ'),
 (4.78743961352657, 'B00IX1I3G6'),
 (4.791666666666667, 'B00A4EK4CQ'),
 (4.795454545454546, 'B00G4IV2VI'),
 (4.8, 'B004KNWWP4'),
 (4.8, 'B004KNWWWM'),
 (4.8, 'B004KNWX1M'),
 (4.8, 'B004WKPW0W'),
 (4.8, 'B005EISPLE'),
 (4.8, 'B005EISPOG')]
```

In [28]:
```
#verified_purchase is 6395. unverified purchase is 520.
```

In [29]:
```
verifiedcounts=defaultdict(int)
verifiedcounts
for d in dataset:
  verifiedcounts[d['verified_purchase']]+=1
verifiedcounts
```

Out[29]:
```
defaultdict(int, {})
```

Out[29]:
```
defaultdict(int, {False: 520, True: 6395})
```

In [30]:
```python
dataset[0]
len(dataset)
```

Out[30]:
```
{'customer_id': '48872127',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'BT00CTOYC0',
 'product_parent': '506740729',
 'product_title': 'Amazon.com $15 Gift Card in a Greeting Card (Amazon
Surprise Box Design)',
 'review_body': 'I love that I have instant, helpful options when I fo
rget a birthday! Thanks for saving the day Amazon!',
 'review_date': '2015-08-31',
 'review_headline': 'Quick Solution for Forgotten Occasion',
 'review_id': 'RVN4P3RU4F8IE',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

Out[30]:
```
6915
```

In [31]:
```python
'''plotted a Bar chart between verified purchase and
unverified purchase.verified purchase is 6395. unverified purchase is
520'''
```

Out[31]:
```
'plotted a Bar chart between verified purchase and \nunverified purcha
se.verified purchase is 6395. unverified purchase is 520'
```

In [32]:
```
unverified_purchase=sum([d['verified_purchase']==False  for d in data
set ])
unverified_purchase
print('\n')
verified_purchase=sum([d['verified_purchase']== True for d in dataset
])
verified_purchase
index=[1]
p1=plt.bar(index,verified_purchase,color='green')
index=[2]
p2=plt.bar(index,unverified_purchase,color='red')
plt.gca().set(title='verified vs unverified purchase',ylabel='total n
umber of purchase',xlabel='purchase category')
plt.xticks([])
plt.legend((p1[0],p2[0]),('verified_purchase','unverified_purchase'))
plt.show()
```

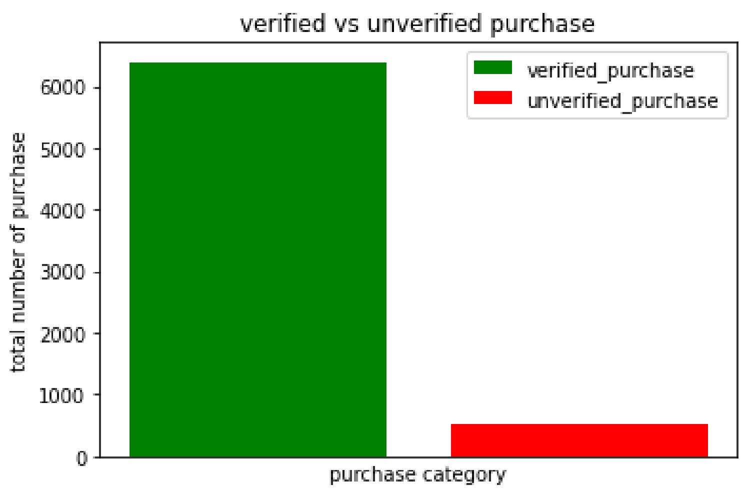
Out[32]: 520

Out[32]: 6395

Out[32]:
```
[Text(0, 0.5, 'total number of purchase'),
 Text(0.5, 0, 'purchase category'),
 Text(0.5, 1.0, 'verified vs unverified purchase')]
```

Out[32]: ([], <a list of 0 Text major ticklabel objects>)

Out[32]: <matplotlib.legend.Legend at 0x7f0156665048>



In [33]:
```
productcounts=defaultdict(int)
for d in dataset:
  productcounts[d['product_id']]+=1
```

In [34]:
```
counts=[(productcounts[p],p) for p in productcounts]
counts.sort()
counts[-10:]
```

Out[34]:
```
[(118, 'B004LLIKY2'),
 (134, 'BT00CTOUNS'),
 (148, 'B007V6EVY2'),
 (152, 'B00A48G0D4'),
 (152, 'BT00DDC7CE'),
 (154, 'B0091JKU5Q'),
 (167, 'B004KNWWO0'),
 (207, 'B00IX1I3G6'),
 (236, 'BT00DDVMVQ'),
 (508, 'B004LLIKVU')]
```

In [35]:
```
nRatings=len(dataset)
nRatings
```

Out[35]:
```
6915
```

In [36]:
```
average=0
for d in dataset:
  average+=d['star_rating']
average/=nRatings
average
```

Out[36]:
```
4.806073752711497
```

In [37]:
```
#total customer headcount is 3801 and products count is 848.
```

In [38]:
```
users=set()
items=set()
for d in dataset:
  users.add(d['customer_id'])
  items.add(d['product_id'])
len(users),len(items)
```

Out[38]:
```
(3801, 848)
```

In [39]:
```
dataset[0]
```

Out[39]:
```
{'customer_id': '48872127',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'BT00CTOYC0',
 'product_parent': '506740729',
 'product_title': 'Amazon.com $15 Gift Card in a Greeting Card (Amazon
Surprise Box Design)',
 'review_body': 'I love that I have instant, helpful options when I fo
rget a birthday! Thanks for saving the day Amazon!',
 'review_date': '2015-08-31',
 'review_headline': 'Quick Solution for Forgotten Occasion',
 'review_id': 'RVN4P3RU4F8IE',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

In [40]:
```
avverified=0
avunverified=0
nverified=0
nunverified=0
for d in dataset:
  if d['verified_purchase']==True:
    avverified+=d['star_rating']
    nverified+=1
  else:
    avunverified+=d['star_rating']
    nunverified+=1
avverified/=nverified
avunverified/=nunverified
avverified,avunverified
```

Out[40]:
```
(4.8151681000781865, 4.694230769230769)
```

In [41]:
```
'''Average for Verified rating is 4.8151.
Average for unverified rating also is somewhere nearby i.e. 4.694'''
```

Out[41]:
```
'Average for Verified rating is 4.8151. \nAverage for unverified ratin
g also is somewhere nearby i.e. 4.694'
```

In [42]:
```
verifiedRatings=[d['star_rating'] for d in dataset
                 if d['verified_purchase']==True ]
unverifiedRatings=[d['star_rating'] for d in dataset
                 if d['verified_purchase']==False ]
sum(verifiedRatings)/len(verifiedRatings)
print('\n')
sum(unverifiedRatings)/len(unverifiedRatings)
dataset[0]
```

Out[42]: 4.8151681000781865

Out[42]: 4.694230769230769

Out[42]:
```
{'customer_id': '48872127',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'BT00CTOYC0',
 'product_parent': '506740729',
 'product_title': 'Amazon.com $15 Gift Card in a Greeting Card (Amazon
Surprise Box Design)',
 'review_body': 'I love that I have instant, helpful options when I fo
rget a birthday! Thanks for saving the day Amazon!',
 'review_date': '2015-08-31',
 'review_headline': 'Quick Solution for Forgotten Occasion',
 'review_id': 'RVN4P3RU4F8IE',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2015}
```

In [43]:
```
from collections import defaultdict
wordcount=defaultdict(int)
for d in dataset:
  for w in d['review_body'].split():
    wordcount[w]+=1
print(len(wordcount))
```

```
15964
```

In [44]:
```
wordcount=defaultdict(int)
import string
for d in dataset:
  r="".join([c for c in d['review_body'].lower() if c not in string.p
unctuation])
  for w in r.split():
    wordcount[w]+=1
print(len(wordcount))
```

```
8486
```

In [45]:
```
counts=[(wordcount[w],w) for w in wordcount]
counts.sort()
counts.reverse()
words=[x[1] for x in counts[:1000]]
wordid=dict(zip(words,range(len(words))))
wordset=set(words)
print(len(wordset))
```

```
1000
```

In [105]:
```
'''Given the feature function and our counts vector, we will Define o
ur x vector for the  Regression model)
we will Fit our model using a Ridge Model with (arying alpha values a
nd  fit_intercept = False).
Using our model, we will Make our Predictions.
then Find the MSE between our predictions and our y vector.'''
```

Out[105]:
```
'Given the feature function and our counts vector, we will Define our
x vector for the  Regression model)\nwe will Fit our model using a Rid
ge Model with (arying alpha values and  fit_intercept = False).\nUsing
our model, we will Make our Predictions.\nthen Find the MSE between ou
r predictions and our y vector.'
```

In [46]:
```
import string
def feature(datum):
  feat=[0]*len(words)
  r=''.join([c for c in datum['review_body'].lower() if not c in stri
ng.punctuation])
  for w in r.split():
    if w in words:
      feat[wordid[w]]+=1
  feat.append(1)
  return feat
```

In [47]:
```
import random
import numpy
random.shuffle(dataset)
x=[feature(d) for d in dataset]
y=[d['star_rating'] for d in dataset]
y[-10:]
```

Out[47]:
```
[5, 5, 5, 1, 5, 5, 5, 5, 5, 5]
```

In [48]:
```
N=len(x)
x_train=x[:N//2]
x_valid=x[N//2:3*N//4]
x_test=x[3*N//4:]
y_train=y[:N//2]
y_valid=y[N//2:3*N//4]
y_test=y[3*N//4:]
```

In [49]:
```
print(len(x))
print(len(x_train))
print(len(x_valid))
print(len(x_test))
```

```
6915
3457
1729
1729
```

In [50]:
```
def MSE(model,x,y):
  predictions=model.predict(x)
  differences=[(a-b)**2 for (a,b) in zip(predictions,y)]
  return sum(differences)/len(differences)
```

In [51]:
```
bestModel=None
bestMSE=None
from sklearn import linear_model
for lamb in [0.01,0.1,1,10,100]:
  model=linear_model.Ridge(lamb,fit_intercept=False)
  model.fit(x_train,y_train)
  mseTrain=MSE(model,x_train,y_train)
  msevalid=MSE(model,x_valid,y_valid)
  mseTrain=MSE(model,x_train,y_train)
  mseTrain=MSE(model,x_train,y_train)
  print("lambda="+str(lamb)+",training/validation error="+str(mseTrai
n)+'\n'+str(msevalid))
  if not bestModel or msevalid<bestMSE:
    bestModel=model
    bestMSE=msevalid
```

Out[51]:
```
Ridge(alpha=0.01, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

lambda=0.01,training/validation error=0.1573972276869462
0.6678151909540279
```

Out[51]:
```
Ridge(alpha=0.1, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

lambda=0.1,training/validation error=0.1574437260048804
0.6510534590872811
```

Out[51]:
```
Ridge(alpha=1, copy_X=True, fit_intercept=False, max_iter=None, normal
ize=False,
      random_state=None, solver='auto', tol=0.001)

lambda=1,training/validation error=0.1597390661715712
0.5514301721223738
```

Out[51]:
```
Ridge(alpha=10, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

lambda=10,training/validation error=0.18625713051791296
0.4009840449500755
```

Out[51]:
```
Ridge(alpha=100, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)

lambda=100,training/validation error=0.30989609196594287
0.4204616182839623
```

In [103]:
```
#using the Ridge model our best MSE stands out to be 0.40098
```

In [52]:
```
print(bestModel)
print(bestMSE)
```

```
Ridge(alpha=10, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
0.4009840449500755
```

In [ ]:

In [53]:
```
import pandas as pd
df=pd.DataFrame(dataset)
df.head(5)
```

Out[53]:

	marketplace	customer_id	review_id	product_id	product_parer
0	US	20852464	R1Z9IP4ZU76ZCL	B00A48G0D4	84870327
1	US	14842411	R27P0OZG1NZGA4	B004LLILM8	75924939
2	US	25660174	R3SM6M5KU4R2UL	B0091JKU5Q	34028330
3	US	11574633	RFDJKUBRQOS34	B004KNWX26	31826993
4	US	44115816	R5ZDD7WOXFG5T	B004KNWX4O	55384427

In [54]:
```
df.columns
```

Out[54]:
```
Index(['marketplace', 'customer_id', 'review_id', 'product_id',
       'product_parent', 'product_title', 'product_category', 'star_ra
ting',
       'helpful_votes', 'total_votes', 'vine', 'verified_purchase',
       'review_headline', 'review_body', 'review_date', 'yearint'],
      dtype='object')
```

In [55]:
```
df.shape
```

Out[55]:
```
(6915, 16)
```

In [56]:
```
'''star_rating,helpful_votes,total_votes,
yearint are scale variables,rest all are categorical variables'''
```

Out[56]:
```
'star_rating,helpful_votes,total_votes,\nyearint are scale variables,r
est all are categorical variables'
```

In [57]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6915 entries, 0 to 6914
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   marketplace        6915 non-null   object
 1   customer_id        6915 non-null   object
 2   review_id          6915 non-null   object
 3   product_id         6915 non-null   object
 4   product_parent     6915 non-null   object
 5   product_title      6915 non-null   object
 6   product_category   6915 non-null   object
 7   star_rating        6915 non-null   int64 
 8   helpful_votes      6915 non-null   int64 
 9   total_votes        6915 non-null   int64 
 10  vine               6915 non-null   bool  
 11  verified_purchase  6915 non-null   bool  
 12  review_headline    6915 non-null   object
 13  review_body        6915 non-null   object
 14  review_date        6915 non-null   object
 15  yearint            6915 non-null   int64 
dtypes: bool(2), int64(4), object(10)
memory usage: 770.0+ KB
```

In [58]:
```
'''here we got more descriptions of the input variable
and those descriptions are like getting mean,std deviation,min,max,an
d data in
25%,50% and 75% quartile for all the listed input features'''
```

Out[58]:
```
'here we got more descriptions of the input variable \nand those descr
iptions are like getting mean,std deviation,min,max,and data in \n25%,
50% and 75% quartile for all the listed input features'
```

In [59]:
```
df.describe()
```

Out[59]:

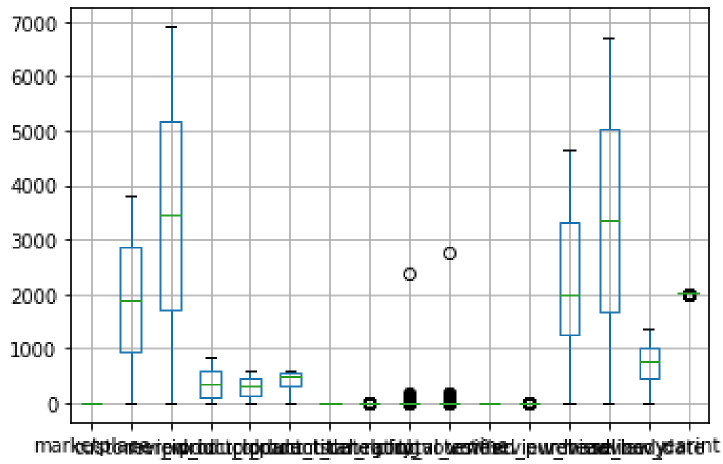
	star_rating	helpful_votes	total_votes	yearint
count	6915.000000	6915.000000	6915.000000	6915.000000
mean	4.806074	0.735358	0.861316	2013.421547
std	0.678074	29.120181	33.726331	1.055599
min	1.000000	0.000000	0.000000	2011.000000
25%	5.000000	0.000000	0.000000	2013.000000
50%	5.000000	0.000000	0.000000	2013.000000
75%	5.000000	0.000000	0.000000	2014.000000
max	5.000000	2383.000000	2763.000000	2015.000000

In [60]:
```
df.isnull().sum()
```
Out[60]:
```
marketplace          0
customer_id          0
review_id            0
product_id           0
product_parent       0
product_title        0
product_category     0
star_rating          0
helpful_votes        0
total_votes          0
vine                 0
verified_purchase    0
review_headline      0
review_body          0
review_date          0
yearint              0
dtype: int64
```
In [61]:
```
#here for all categorical variables we convert bool values to equivalent 1 and 0.
```
In [62]:
```
#DUMMY CODING USING THE LOOP STRUCTURE
for col in df.columns:
  if df[col].dtype=='object':
    df[col]=pd.Categorical(df[col]).codes
df.head(5)
```
Out[62]:

	marketplace	customer_id	review_id	product_id	product_parent	product_ti
0	0	1125	1771	395	476	3
1	0	583	2216	119	418	4
2	0	1472	5101	336	147	5
3	0	173	5903	47	131	2
4	0	2766	5422	52	291	1

In [63]:
```
#Here Box plot is plotted
```

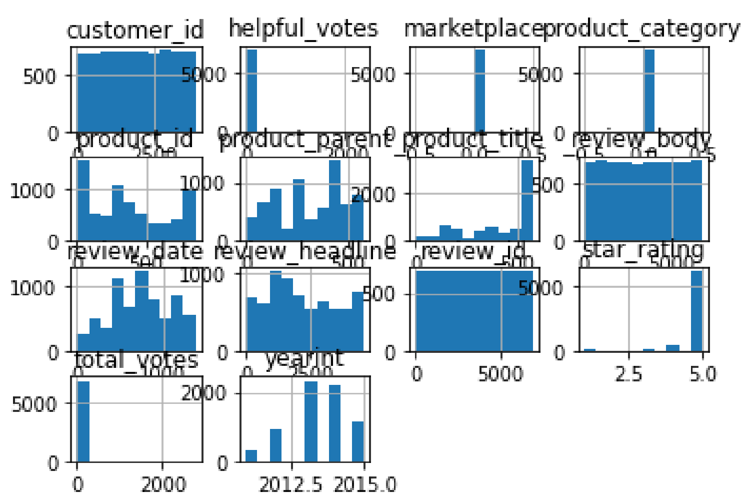
In [64]:
```
df.boxplot()
```
Out[64]: <matplotlib.axes._subplots.AxesSubplot at 0x7f01553b6e10>



In [65]:
```
#here histogram is being plotted to draw comparisons between variable
s
```

In [66]:
```
x=df.drop(['vine','verified_purchase'],axis=1)
x.hist(grid='off')
```

Out[66]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f01551a686
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0155168c8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f015512bfd
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0155075f6
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f015504040
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f01550054e
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154fcb39
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154f900f
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0154f9019
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154f2b0b
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154e6fb0
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154e36be
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0154dfccc
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154dc3b7
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154d8d63
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0154d5b43
8>]],
      dtype=object)
```



In [67]:
```
'''Heat map plotted to evaluate
correlation between the variables'''
```

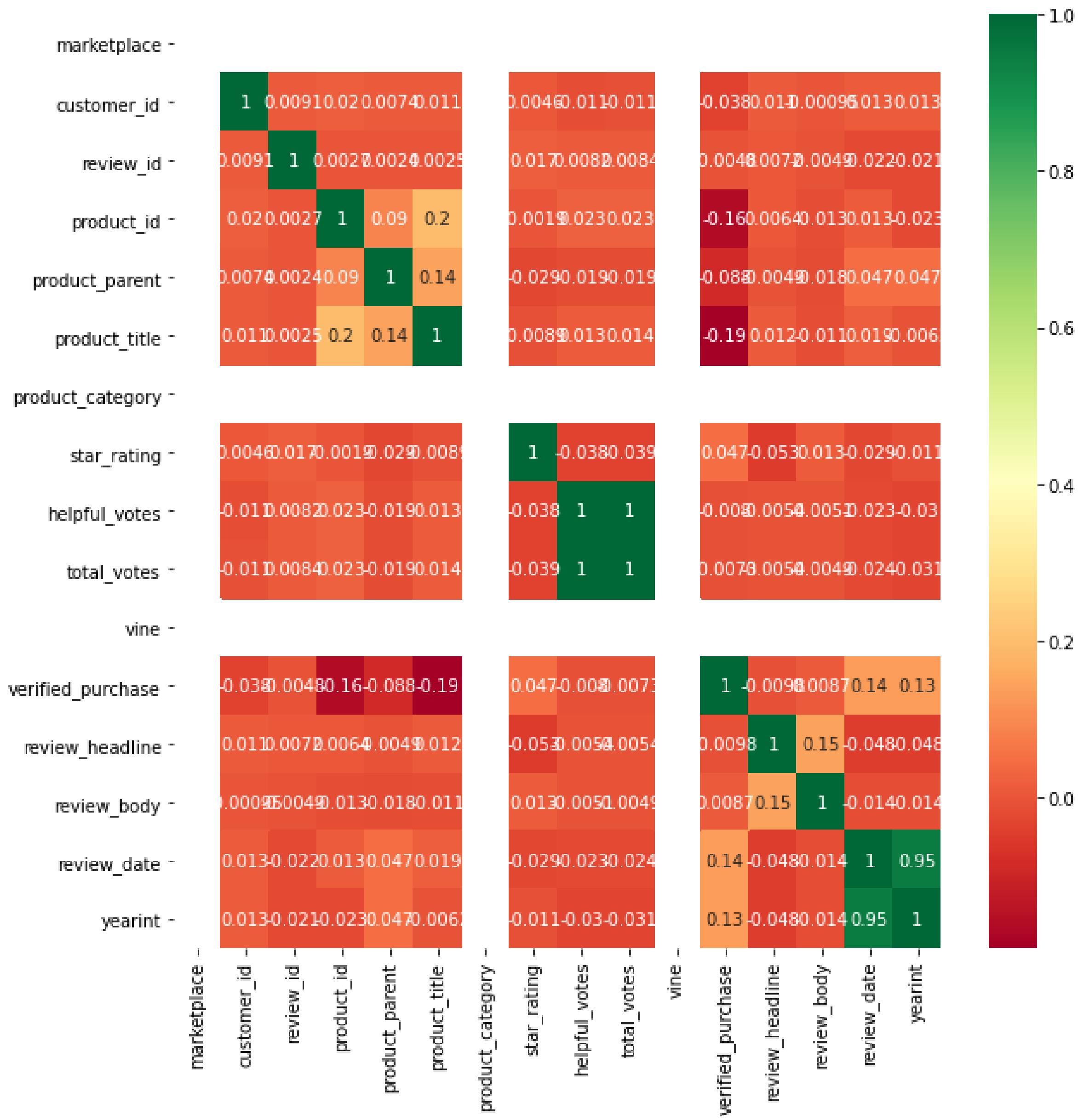
Out[67]:
```
'Heat map plotted to evaluate\ncorrelation between the variables'
```

In [68]:
```
x=df.corr()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
plt.subplots(figsize=(10,10))
sns.heatmap(x,cmap='RdYlGn',annot=True)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:1
9: FutureWarning: pandas.util.testing is deprecated. Use the functions
in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[68]:
```
(<Figure size 720x720 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f0154e82908>)
```

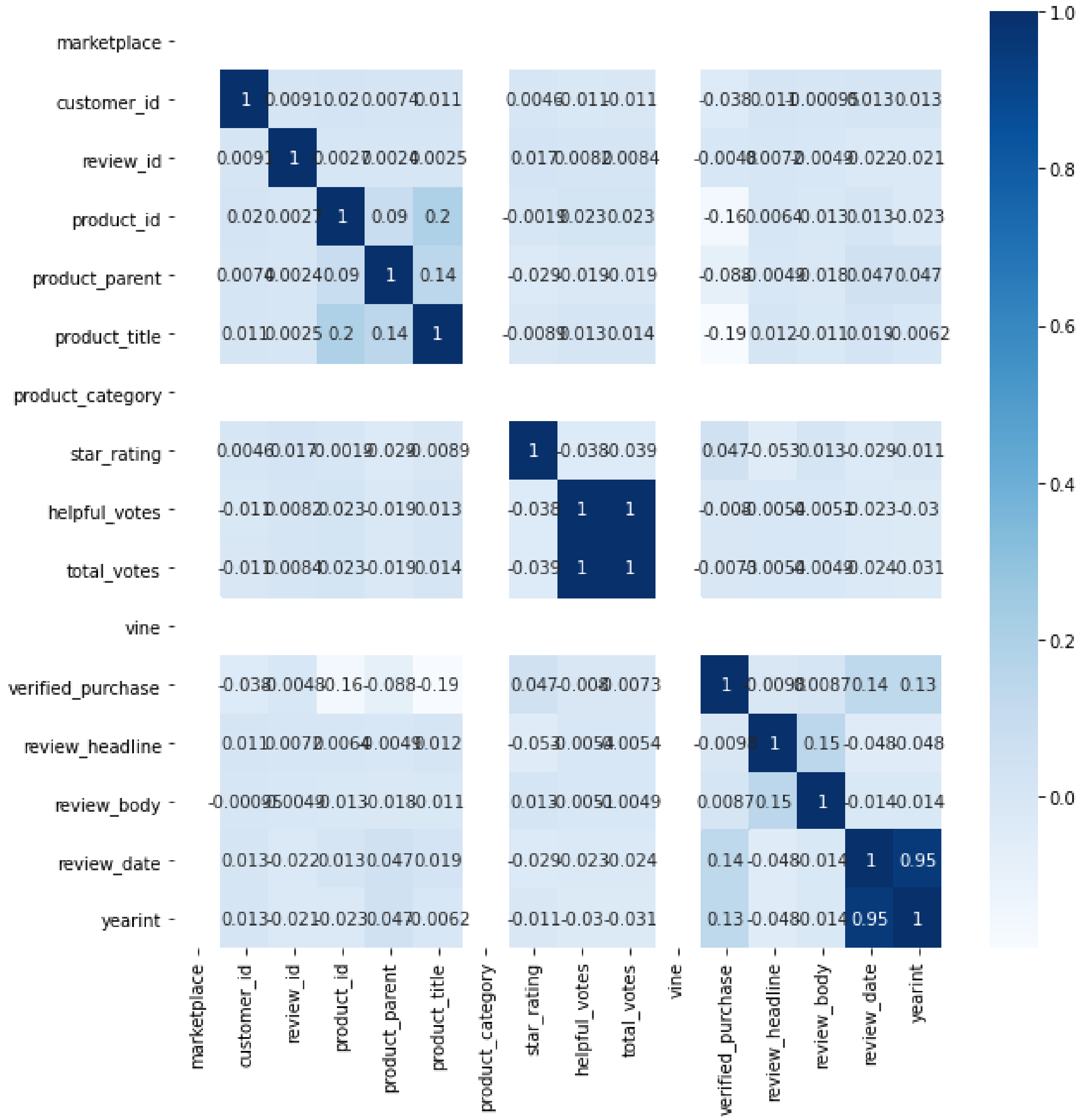
Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0154e82908>
```



In [69]:
```python
plt.subplots(figsize=(10,10))
sns.heatmap(x,cmap='Blues',annot=True)
plt.show()
```

Out[69]:
```
(<Figure size 720x720 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7f01546d8160>)
```

Out[69]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01546d8160>
```



In [70]:
```python
#useful data structures
userperitem=defaultdict(set)
itemsperuser=defaultdict(set)
itemnames={}
```

In [71]:
```python
for d in dataset:
  user,item=d['customer_id'],d['product_id']
  usersperitem[item].add(user)
  itemsperuser[user].add(item)
  itemnames[item]=d['product_title']
```

In [72]:
```
'''we want a recommendation function that return items
similar to a candidate item i,our strategy is as follows:
find the set of users who purchased i
iterate over all other items other than i
for all other items compute their
similiarity with i and store it.
sort all other items by jaccard similiarity
return the most similar.'''
```

Out[72]:
```
'we want a recommendation function that return items\nsimilar to a can
didate item i,our strategy is as follows:\nfind the set of users who p
urchased i\niterate over all other items other than i\nfor all other i
tems compute their \nsimiliarity with i and store it.\nsort all other
items by jaccard similiarity\nreturn the most similar.'
```

In [73]:
```
def jaccard(s1,s2):
  numer=len(s1.intersection(s2))
  denom=len(s1.union(s2))
  return(numer/denom)
def mostsimilar(i):
  similiarities=[]
  users=usersperitem[i]
  for l2 in usersperitem:
    if l2==i:continue
    sim=jaccard(users,usersperitem[l2])
    similiarities.append((sim,l2))
  return similiarities[:10]
```

In [74]:
```
query=dataset[6]['product_id']
query
```

Out[74]:
```
'BT00DDVMVQ'
```

In [75]:
```
mostsimilar(query)
```

Out[75]:
```
[(0.0, 'B00A48G0D4'),
 (0.0, 'B004LLILM8'),
 (0.005181347150259676, 'B0091JKU5Q'),
 (0.0, 'B004KNWX26'),
 (0.0, 'B004KNWX4O'),
 (0.0035842293906810036, 'B00G4IWEZG'),
 (0.009153318077803204, 'B00IX1I3G6'),
 (0.005797101449275362, 'B0091JKY0M'),
 (0.0, 'B0066AZGD4'),
 (0.029649595687331536, 'B007V6EVY2')]
```

In [76]:
```
itemnames[query]
```

Out[76]:
```
'Amazon eGift Card - Smile'
```

In [77]:
```
[itemnames[x[1]] for x in mostsimilar(query)]
```

Out[77]:
```
['Amazon eGift Card - Happy Birthday (Candles)',
 "Amazon eGift Card - Happy Mother's Day (Butterflies)",
 'Amazon.com Gift Card for Any Amount in a Snowflake Tin (Happy Holida
ys Card Design)',
 'Amazon Gift Card - Print - Merry Christmas (Shopping Snowman)',
 "Amazon Gift Card - Print - Happy Mother's Day (Butterflies)",
 'Amazon Gift Card - Print - Merry Christmas (Pine)',
 'Amazon.com Gift Card Balance Reload',
 'Amazon.com Gift Card for Any Amount in a Santa Tin (Ho! Ho! Ho! Card
Design)',
 'Amazon eGift Card - Upload Your Photo - Gift for You',
 'Amazon Gift Card - Print - Happy Birthday (Presents)']
```

In [78]:
```
'''it is sufficient to iterate over thoose
items purchased by one of the users
who purchased i.
find the set of users who purchased i.
iterate over all users who purchased i
build a condidate set from all items
those users consumed.for items in this set,
compute their similiarity with i and store it.
sort all other items by jaccard similiarity
return the most similar'''
```

Out[78]:
```
'it is sufficient to iterate over thoose\nitems purchased by one of th
e users \nwho purchased i.\nfind the set of users who purchased i.\nit
erate over all users who purchased i\nbuild a condidate set from all i
tems \nthose users consumed.for items in this set,\ncompute their simi
liarity with i and store it.\nsort all other items by jaccard similiar
ity\nreturn the most similar'
```

In [79]:
```
def mostsimilarfast(i):
  similiarities=[]
  users=usersperitem[i]
  candidateitems=set()
  for u in users:
    candidateitems=candidateitems.union(itemsperuser[u])
  for l2 in candidateitems:
    if l2==i:
      continue
    sim=jaccard(users,usersperitem[l2])
    similiarities.append((sim,l2))
  similiarities.sort(reverse=True)
  return similiarities[:10]
```

In [80]:
```
query=dataset[2]['product_id']
mostsimilarfast(query)
```

Out[80]:
```
[(0.24766355140186916, 'B0091JKY0M'),
 (0.07692307692307693, 'B00CHQ7ESQ'),
 (0.057803468208092484, 'B0091JKLN2'),
 (0.03296703296703297, 'B0091JKFG0'),
 (0.026881720430107527, 'B0080IR4MQ'),
 (0.022988505747126436, 'B0091JKYLQ'),
 (0.02127659574468085, 'B005ISQ62U'),
 (0.020942408376963352, 'B005ESMF5G'),
 (0.020833333333333332, 'B007RFEL42'),
 (0.01764705882352941, 'B00JDQJVF2')]
```

In [81]:
```
'''The user(u)'s rating for an item i is a
weighted combination of all of their
previous ratings for item j.
the weight for each rating is given by
the jaccard similiarity between i and j.'''
```

Out[81]:
```
"The user(u)'s rating for an item i is a \nweighted combination of all
of their\nprevious ratings for item j.\nthe weight for each rating is
given by\nthe jaccard similiarity between i and j."
```

In [82]:
```
#more utility dta structures
reviewsperuser=defaultdict(list)
reviewsperitem=defaultdict(list)
for d in dataset:
  user,item=d['customer_id'],d['product_id']
  reviewsperuser[user].append(d)
  reviewsperitem[item].append(d)
```

In [83]:
```
ratingmean=sum([d['star_rating'] for d in dataset])/len(dataset)
ratingmean
```

Out[83]:
```
4.806073752711497
```

In [84]:
```
def predictrating(user,item):
  ratings=[]
  similiarities=[]
  for d in reviewsperuser[user]:
    i2=d['product_id']
    if i2==item:continue
    ratings.append(d['star_rating'])
    similiarities.append(d['star_rating'])
    similiarities.append(jaccard(userspeitem[item],userspeitem[i2
]))
  if sum(similiarities)>0:
    weightedratings=[(x*y) for x,y in zip(ratings,similiarities)]
    return sum(weightedratings)/sum(similiarities)
  else:
    return ratingmean
```

In [85]:
```
dataset[1]
```

Out[85]:
```
{'customer_id': '14842411',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B004LLILM8',
 'product_parent': '759249391',
 'product_title': "Amazon eGift Card - Happy Mother's Day (Butterflie
s)",
 'review_body': 'This is the perfect gift to the perfect store with th
e best prices. How can you ask for more? Amazon gift cards are easy to
use and work for everyone.',
 'review_date': '2014-02-28',
 'review_headline': 'Amazon gift card',
 'review_id': 'R27P0OZG1NZGA4',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2014}
```

In [86]:
```
u,i=dataset[0]['customer_id'],dataset[0]['product_id']
predictrating(u,i)
```

Out[86]:
```
4.806073752711497
```

In [87]:
```
def MSE(predictions,labels):
  differences=[(x-y)**2 for x,y in zip(predictions,labels)]
  return sum(differences)/len(differences)
```

In [88]:
```
alwayspredictmean=[ratingmean for d in dataset ]
cpredictions=[predictrating(d['customer_id'],d['product_id']) for d i
n dataset]
```

In [89]:
```
'''here MSE doing worse than in case of always predicting the
mean which is 0.4597 compared to 0.9008.we can try different other
techniques like similiarity based on users
rather than items or a different weighting scheme.
still we are are able to demonstrate 2 different recommender
systems over here based on jaccard similiarity as such.'''
```

Out[89]:
```
'here MSE doing worse than in case of always predicting the \nmean whi
ch is 0.4597 compared to 0.9008.we can try different other\ntechniques
like similiarity based on users \nrather than items or a different wei
ghting scheme.\nstill we are are able to demonstrate 2 different recom
mender \nsystems over here based on jaccard similiarity as such.'
```

In [90]:
```
labels=[d['star_rating']for d in dataset]
MSE(alwayspredictmean,labels)
```

Out[90]:
```
0.4597172671562807
```

In [91]:
```
labels=[d['star_rating']for d in dataset]
MSE(cpredictions,labels)
```

Out[91]:
```
0.9044472498628396
```

In [102]:
```
'''since MSE is doing worse we can try different other
techniques like  a different weighting scheme.let us Build Latent FAC
TOR MODELS.'''
```

Out[102]:
```
'since MSE is doing worse we can try different other\ntechniques like
a different weighting scheme.let us Build Latent FACTOR MODELS.'
```

In [92]:
```
#coding for latent factor models
N=len(dataset)
nusers=len(reviewsperuser)
nitems=len(reviewsperitem)
users=list(reviewsperuser.keys())
items=list(reviewsperitem.keys())
```

In [93]:
```
alpha=ratingmean
userbiases=defaultdict(float)
itembiases=defaultdict(float)
```

In [94]:
```
def prediction(user,item):
  return alpha+userbiases[user]+itembiases[item]
```

In [95]:
```
'''the gradient descent library we will use expects
a single vector of parameters(theta)which we have to unpack
to produce alpha and beta'''
```

Out[95]:
```
'the gradient descent library we will use expects\na single vector of
parameters(theta)which we have to unpack \nto produce alpha and beta'
```

In [96]:
```
def unpack(theta):
  global alpha
  global userbiases
  global itembiases
  alpha=theta[0]
  userbiases=dict(zip(users,theta[1:nusers+1]))
  itembiases=dict(zip(items,theta[1+nusers:]))
```

In [97]:
```
'''the next function just implement the full cost function
which is required by the gradient descent library'''
def cost(theta,lbels,lamb):
  unpack(theta)
  predictions=[prediction(d['customer_id'],d['product_id'])for d in d
ataset]
  cost=MSE(predictions,labels)
  print("MSE= "+str(cost))
  for u in userbiases:
    cost+=lamb*userbiases[u]**2
  for i in itembiases:
    cost+=lamb*itembiases[i]**2
  return(cost)
```

Out[97]:
```
'the next function just implement the full cost function \nwhich is re
quired by the gradient descent library'
```

In [98]:
```
'''next we implement the derivative function which has a
corresponding derivative term for each parameter'''
def derivative(theta,labels,lamb):
  unpack(theta)
  N=len(dataset)
  dalpha=0
  dUserBiases=defaultdict(float)
  dItemBiases=defaultdict(float)
  for d in dataset:
    u,i=d['customer_id'],d['product_id']
    pred=prediction(u,i)
    diff=pred-d['star_rating']
    dalpha+=2/N*diff
    dUserBiases[u]+=2/N*diff
    dItemBiases[i]+=2/N*diff
  for u in userbiases:
    dUserBiases[u]+=2*lamb*userbiases[u]
  for i in itembiases:
    dItemBiases[i]=2*lamb*itembiases[i]
  dtheta=[dalpha]+[dUserBiases[u] for u in users]+[ dItemBiases[i] fo
r i in items]
  return numpy.array(dtheta)
```

Out[98]:
```
'next we implement the derivative function which has a \ncorresponding
derivative term for each parameter'
```

In [99]:
```
MSE(alwayspredictmean,labels)
```

Out[99]:
```
0.4597172671562807
```

In [101]:
```
'''the gradient descent library we use is called lbfgs
this is a general purpose gradient descent algorithm
which simply requires that we provide a cost function f(x)
and a dervative function f'(x).it can be relatively
straightforwardly adapted to other gradient descent problems'''
import scipy.optimize
scipy.optimize.fmin_l_bfgs_b(cost,[ratingmean]+[0.0]*(nusers+nitems),
derivative,args=(labels,0.001))
```

Out[101]:
```
"the gradient descent library we use is called lbfgs\nthis is a genera
l purpose gradient descent algorithm \nwhich simply requires that we p
rovide a cost function f(x)\nand a dervative function f'(x).it can be
relatively\nstraightforwardly adapted to other gradient descent proble
ms"
```

```
MSE= 0.4597172671562807
MSE= 0.4399841504493683
MSE= 0.33694235460752264
MSE= 0.3498790824449461
MSE= 0.3050958667701618
MSE= 0.3217013229905716
MSE= 0.3216735031071555
MSE= 0.32146193553595687
MSE= 0.3213795546185837
MSE= 0.32147743255476496
MSE= 0.3216170385826815
MSE= 0.32166921365178175
MSE= 0.32167877042435505
MSE= 0.32169211206311366
MSE= 0.3217010442175585
MSE= 0.32168309419318064
MSE= 0.3217319774826034
MSE= 0.32168913560084916
```

Out[101]:
```
(array([4.80599347, 0.02449266, 0.04351649, ..., 0.        , 0.
,
        0.        ]),
 0.38201766857060343,
 {'funcalls': 18,
  'grad': array([ 2.64388323e-06, -4.25635507e-08, -1.83714762e-08,
...,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]),
  'nit': 15,
  'task': b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'warnflag': 0})
```

In [107]:
```
'''here we are able to optimize using scipy.optimize and our MSE has
 improved to  0.32168'''
```

Out[107]:
```
'here we are able to optimize using scipy.optimize and our MSE has imp
roved to  0.32168'
```