Project Submission

This notebook will be your project submission. All tasks will be listed in the order of the Courses that they appear in. The tasks will be the same as in the Capstone Example Notebook, but in this submission you ***MUST*** use another dataset. Failure to do so will result in a large penalty to your grade in this course.

Finding your dataset

Take some time to find an interesting dataset! There is a reading discussing various places where datasets can be found, but if you are able to process it, go ahead and use it! Do note, for some tasks in this project, each entry will need 3+ attributes, so keep that in mind when finding datasets. After you have found your dataset, the tasks will continue as in the Example Notebook. You will be graded based on the tasks and your results. Best of luck!

As Reviewer:

Your job will be to verify the calculations made at each "TODO" labeled throughout the notebook.

First Step: Imports

In the next cell we will give you all of the imports you should need to do your project. Feel free to add more if you would like, but these should be sufficient.

In [ ]:
```
import gzip
import csv
from collections import defaultdict
import random
import numpy
import scipy.optimize
import string
from sklearn import linear_model
from nltk.stem.porter import PorterStemmer # Stemming
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [ ]:
```
'''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[ ]:
```
'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.\nThe review dateset comprises of
many customers'
```

Task 1: Data Processing

TODO 1: Read the data and Fill your dataset

In [ ]:
```
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)
dataset=[d for d in dataset if 'review_date' in d ]
print('\n')
print(len(dataset))
print(dataset[0])
```

```
148309
{'marketplace': 'US', 'customer_id': '24371595', 'review_id': 'R27ZP1F
1CD0C3Y', 'product_id': 'B004LLIL5A', 'product_parent': '346014806',
'product_title': 'Amazon eGift Card - Celebrate', 'product_category':
'Gift Card', 'star_rating': 5, 'helpful_votes': 0, 'total_votes': 0,
'vine': False, 'verified_purchase': True, 'review_headline': 'Five Sta
rs', 'review_body': 'Great birthday gift for a young adult.', 'review_
date': '2015-08-31'}
```

In [ ]:
```
'''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[ ]:
```
"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 [ ]:
```
dataset[0]
len(dataset)
```

Out[ ]:
```
{'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[ ]:
```
148309
```

In [ ]:
```
'''FIRST WE HAVE TO PREPROCESS OUR DATESET TO EXTRACT ONLY THOSE\nENT
RIES CONTAINING A REVIEW DATE FIELD'''
```

Out[ ]:
```
'FIRST WE HAVE TO PREPROCESS OUR DATESET TO EXTRACT ONLY THOSE\nENTRIE
S CONTAINING A REVIEW DATE FIELD'
```

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

Out[ ]: 148309

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

Out[ ]: 'now let us filter old reviews i.e. those before 2010'

In [ ]:
```
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[ ]:
```
{'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[ ]: 146727

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

Out[ ]: 'let us write other list comprehension to exclude reviews with low hel
pful rates'

In [ ]:
```
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[ ]:
```
{'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[ ]:
```
146461
```

In [ ]:
```
'''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[ ]:
```
'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 [ ]: 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[ ]: {'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[ ]: {'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[ ]: 11048
```

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

```
Out[ ]: 'let us remove short reviews which may be uninformative'
```

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

Out[ ]:
```
{'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[ ]:
```
6915
```

In [ ]:
```
dataset1=dataset
import pandas as pd
df=pd.DataFrame(dataset)
df.head(5)
```

Out[ ]:

	marketplace	customer_id	review_id	product_id	product_parent
0	US	48872127	RVN4P3RU4F8IE	BT00CTOYC0	50674072
1	US	25208893	R13UP4ELOFYDB5	B00PG40PAK	750842252
2	US	13376158	R3KLV1HD0EFCSV	B005Z3D5OU	37936893
3	US	47184195	R3SILVKZXUV8TT	B00A4EK4YO	16766865
4	US	1094807	R3U229HF6OOJXQ	B004LLIKVU	473048287

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

Out[ ]:
```
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 [ ]:
```
df.shape
```

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

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

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

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

Out[ ]:

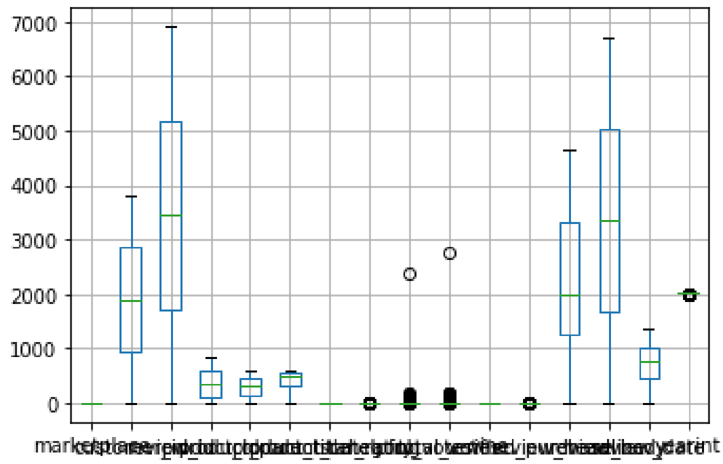
	star_rating	helpful_votes	total_votes	yearint
count	6915.000000	6915.000000	6915.000000	6915.000000
mean	4.806074	0.735358	0.861316	2013.421547
std	0.678074	29.120181	33.726331	1.055599
min	1.000000	0.000000	0.000000	2011.000000
25%	5.000000	0.000000	0.000000	2013.000000
50%	5.000000	0.000000	0.000000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

In [ ]:
```
df.isnull().sum()
```
Out[ ]:
```
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 [ ]:
```
'''here for all categorical variables we convert bool values to equiv
alent 1 and 0.'''
```
Out[ ]:
```
'here for all categorical variables we convert bool values to equivale
nt 1 and 0.'
```
In [ ]:
```
#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[ ]:

	marketplace	customer_id	review_id	product_id	product_parent	product_ti
0	0	3213	6722	775	258	5
1	0	1443	196	735	415	5
2	0	390	4678	250	172	5
3	0	3061	5090	401	44	4
4	0	105	5189	71	236	5

In [ ]:
```
'''Here Box plot is plotted'''
```
Out[ ]:
```
'Here Box plot is plotted'
```

In [ ]:
```
df.boxplot()
```
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc400466e48>
```



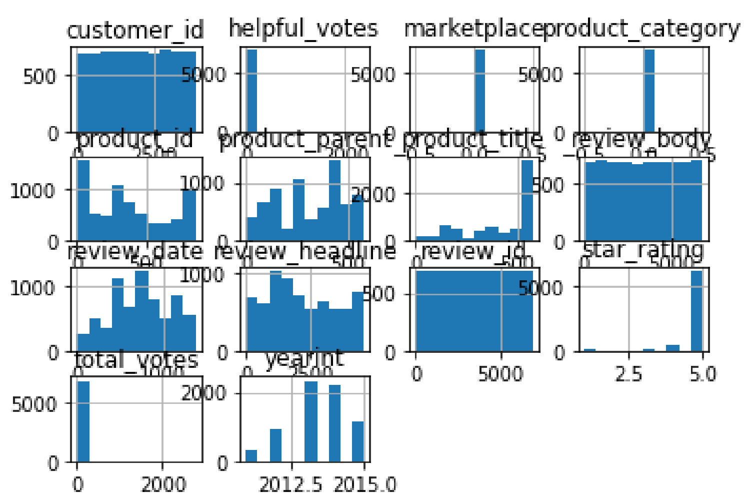
In [ ]:
```
'''here histogram is being plotted to draw comparisons between variab
les'''
```
Out[ ]:
```
'here histogram is being plotted to draw comparisons between variable
s'
```

In [ ]:

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

Out[ ]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3fe4e386
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc40026594
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc4001bf24
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc400148eb
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc40011c89
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc4000ef19
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc400039c5
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc40000f86
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc40000f8d
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3fffa7fd
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffef6c5
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffec755
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffe9235
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffe5f89
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffe1af9
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fc3ffddc78
0>]],
      dtype=object)
```



In [ ]:

```
'''Heat map plotted to evaluate
correlation between the variables'''
```

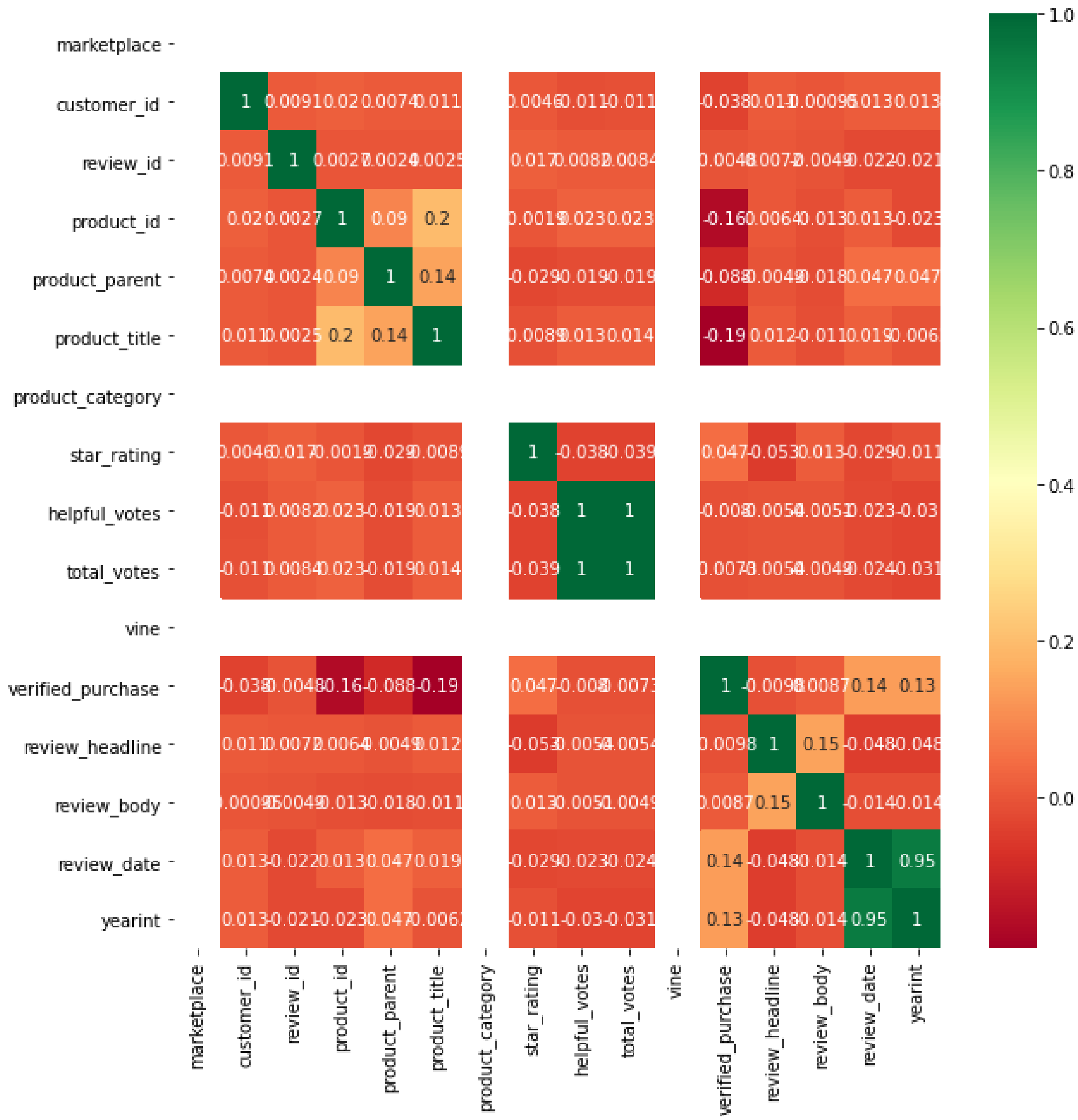
Out[ ]:

```
'Heat map plotted to evaluate\ncorrelation between the variables'
```

In [ ]:
```
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()
```

Out[ ]:
```
(<Figure size 720x720 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fc3ffb8ca58>)
```

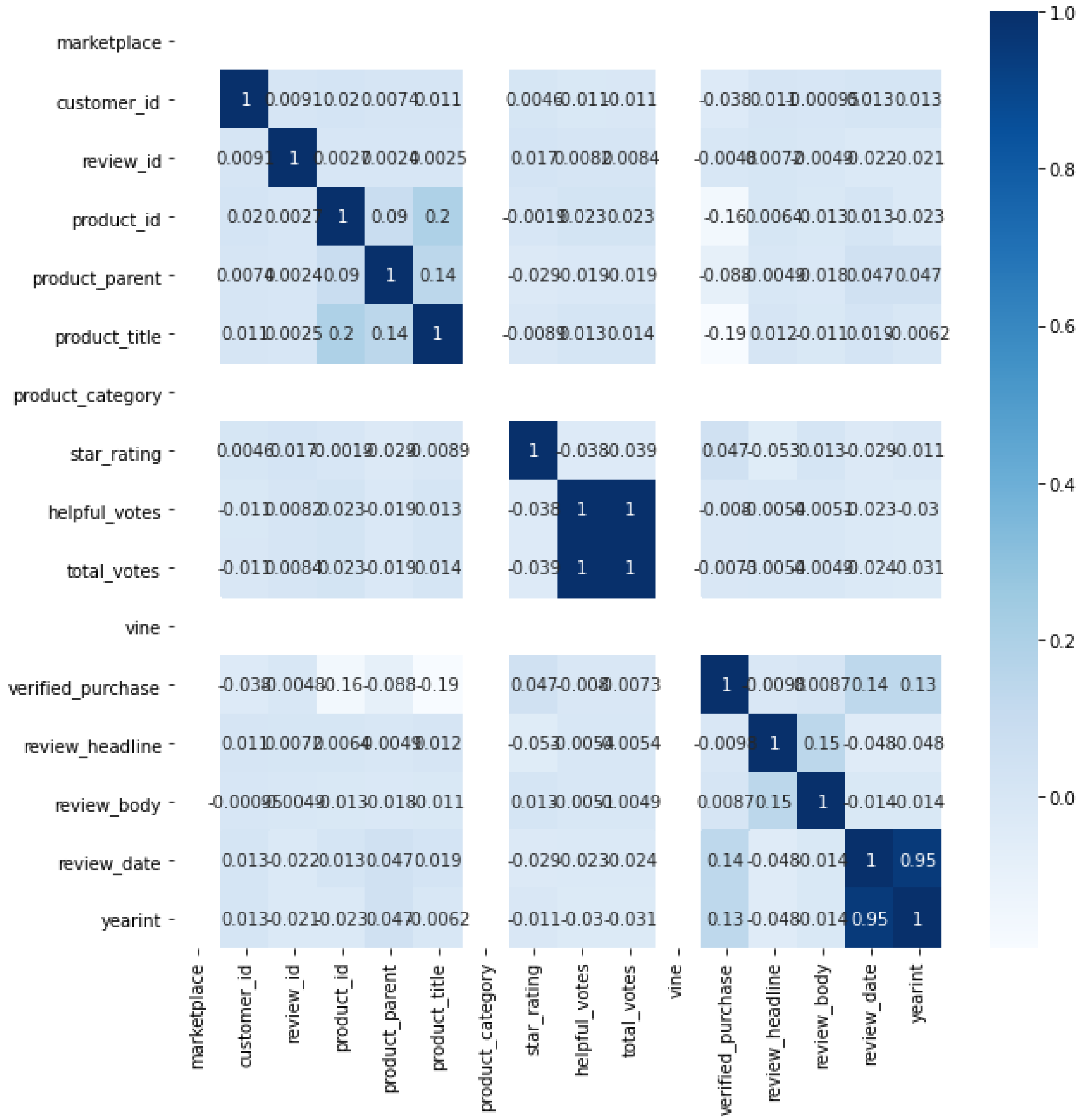
Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc3ffb8ca58>
```



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

```
Out[ ]: (<Figure size 720x720 with 1 Axes>,
         <matplotlib.axes._subplots.AxesSubplot at 0x7fc3ff9fcb70>)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7fc3ff9fcb70>
```



TODO 2: Split the data into a Training and Testing set

First shuffle your data, then split your data. Have Training be the first 80%, and testing be the remaining 20%.

In [ ]:
```
#YOUR CODE HERE
random.shuffle(dataset)
N=len(dataset)
print(len(dataset))
train_len=4*N//5
test_len=N//5
count=0
trainingSet=[]
testSet=[]
```

```
6915
```

In [ ]:
```
for d in dataset:
  count=count+1
  if count<=train_len:
    trainingSet.append(d)
  else:
    testSet.append(d)
print(len(trainingSet), len(testSet))
print("Lengths should be: 5532 1383")
```

```
5532 1383
Lengths should be: 5532 1383
```

Now delete your dataset

You don't want any of your answers to come from your original dataset any longer, but rather your Training Set, this will help you to not make any mistakes later on, especialy when referencing the checkpoint solutions.

In [ ]:
```
#YOUR CODE HERE
del dataset
print("training set sample data")
print(trainingSet[0])
print("test set sample data")
print(testSet[0])
```

```
training set sample data
{'marketplace': 'US', 'customer_id': '43417851', 'review_id': 'R2YQBK3
0N9OUCZ', 'product_id': 'B00A44A3Y0', 'product_parent': '578402716',
'product_title': 'Amazon Gift Card - Print - Happy Birthday (Candle
s)', 'product_category': 'Gift Card', 'star_rating': 5, 'helpful_vote
s': 0, 'total_votes': 0, 'vine': False, 'verified_purchase': True, 're
view_headline': 'You cannot go wrong with a giftcard.', 'review_body':
'I trust the gift card was enjoyed. My son-in-law gets e-books and tha
t makes me believe he enjoyed the gift card.', 'review_date': '2013-06
-07', 'yearint': 2013}
test set sample data
{'marketplace': 'US', 'customer_id': '11102803', 'review_id': 'R339N72
0KVPHHT', 'product_id': 'B00IX1I3G6', 'product_parent': '926539283',
'product_title': 'Amazon.com Gift Card Balance Reload', 'product_categ
ory': 'Gift Card', 'star_rating': 3, 'helpful_votes': 1, 'total_vote
s': 1, 'vine': False, 'verified_purchase': True, 'review_headline': 'L
ook and Wait', 'review_body': 'difficult to find and no verification t
hat I was successful in using online reload but overall OK', 'review_d
ate': '2015-08-26', 'yearint': 2015}
```

TODO 3: Extracting Basic Statistics

Next you need to answer some questions through any means (i.e. write a function or just find the answer) all based on the **Training Set:**

1. How many entries are in your dataset?
2. Pick a non-trivial attribute (i.e. verified purchases in example), what percentage of your data has this atttribute?
3. Pick another different non-trivial attribute, what percentage of your data share both attributes?

In [ ]:
```
#average star rating for the entire dataset comes out as 4.807
ratings=[d['star_rating'] for d in trainingSet]
sum(ratings)/len(ratings)
```

Out[ ]: 4.803687635574837

In [ ]:
```
'''defaultdict"structure from the "collections"library allows us to a
utomate
initializing a dictionary with all zero counts'''
#ratingcounts={1:0,2:0,3:0,4:0,5:0}
ratingcounts=defaultdict(int)
print('\n')
for d in trainingSet:
  ratingcounts[d['star_rating']]+=1
ratingcounts
```

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

Out[ ]:
```
defaultdict(int, {1: 101, 2: 48, 3: 109, 4: 320, 5: 4954})
```

In [ ]:
```
print(ratingcounts[5])
otherratings=ratingcounts[1]+ratingcounts[2]+ratingcounts[3]+ratingco
unts[4]
print(otherratings)
totalratings=otherratings+ratingcounts[5]
```

```
4954
578
```

In [ ]:
```
print('fraction of reviews have 5-star ratings are')
print(ratingcounts[5]/totalratings)
```

```
fraction of reviews have 5-star ratings are
0.8955169920462762
```

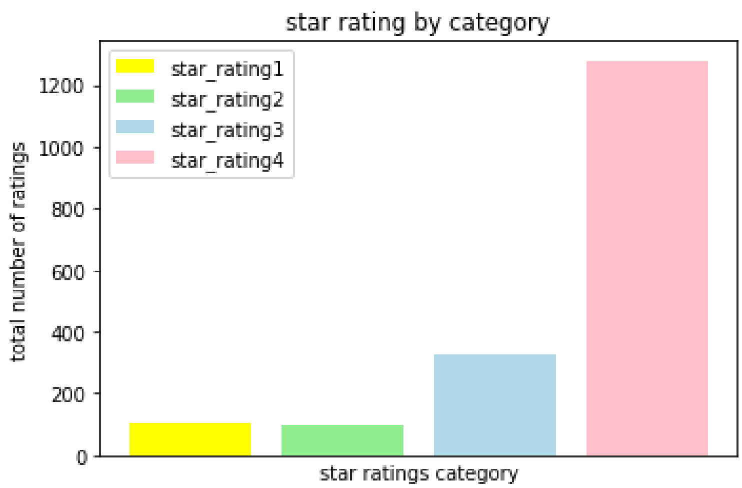
In [ ]:

```
import matplotlib.pyplot as plt
from matplotlib import colors
star1=sum([d['star_rating']for d in trainingSet if d['star_rating'] is 1 ])
star2=sum([d['star_rating']for d in trainingSet if d['star_rating'] is 2 ])
star3=sum([d['star_rating']for d in trainingSet if d['star_rating'] is 3 ])
star4=sum([d['star_rating']for d in trainingSet if d['star_rating'] is 4 ])
star5=sum([d['star_rating']for d in trainingSet 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[ ]: [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[ ]: ([], <a list of 0 Text major ticklabel objects>)

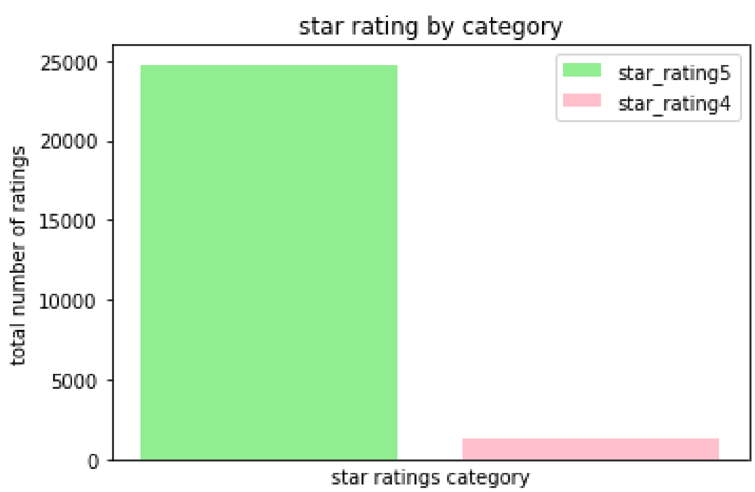
Out[ ]: <matplotlib.legend.Legend at 0x7fc400330a90>



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

Out[ ]: <matplotlib.legend.Legend at 0x7fc400078d68>



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

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

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

Out[ ]: [5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]

In [ ]:
```
'''Average ratings per product stands out to be 4.8367'''
```

Out[ ]: 'Average ratings per product stands out to be 4.8367'

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

Out[ ]: 4.836734693877551

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

Out[ ]:
```
[(4.818181818181818, 'B00BWDH54I'),
 (4.822222222222222, 'B004LLIKY2'),
 (4.825, 'B00AF0K82U'),
 (4.827669902912621, 'B004LLIKVU'),
 (4.833333333333333, 'B004LLILQ4'),
 (4.833333333333333, 'B0062ONA9Q'),
 (4.833333333333333, 'B00CHQ6F9A'),
 (4.833333333333333, 'B00H5BMF00'),
 (4.83453237410072, 'B004KNWWO0'),
 (4.836734693877551, 'B004LLIL5A')]
```

In [ ]:
```
#verified_purchase is 5117. unverified purchase is 415.
```

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

Out[ ]: defaultdict(int, {})

Out[ ]: defaultdict(int, {False: 415, True: 5117})

In [ ]:
```
print('fraction of reviews are from verified purchases is')
print(verifiedcounts[True]/(verifiedcounts[True]+verifiedcounts[False
]))
```

```
fraction of reviews are from verified purchases is
0.9249819233550253
```

In [ ]:
```
trainingSet[0]
len(trainingSet)
```

Out[ ]:
```
{'customer_id': '43417851',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B00A44A3Y0',
 'product_parent': '578402716',
 'product_title': 'Amazon Gift Card - Print - Happy Birthday (Candle
s)',
 'review_body': 'I trust the gift card was enjoyed. My son-in-law gets
e-books and that makes me believe he enjoyed the gift card.',
 'review_date': '2013-06-07',
 'review_headline': 'You cannot go wrong with a giftcard.',
 'review_id': 'R2YQBK30N9OUCZ',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2013}
```

Out[ ]:
```
5532
```

In [ ]:
```
'''plotted a Bar chart between verified purchase and
unverified purchase.verified purchase is 5117.unverified purchase is
 415'''
```

Out[ ]:
```
'plotted a Bar chart between verified purchase and \nunverified purcha
se.verified purchase is 5103.unverified purchase is 429'
```

In [ ]:
```
unverified_purchase=sum([d['verified_purchase']==False  for d in trai
ningSet ])
unverified_purchase
print('\n')
verified_purchase=sum([d['verified_purchase']== True for d in trainin
gSet])
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()
```

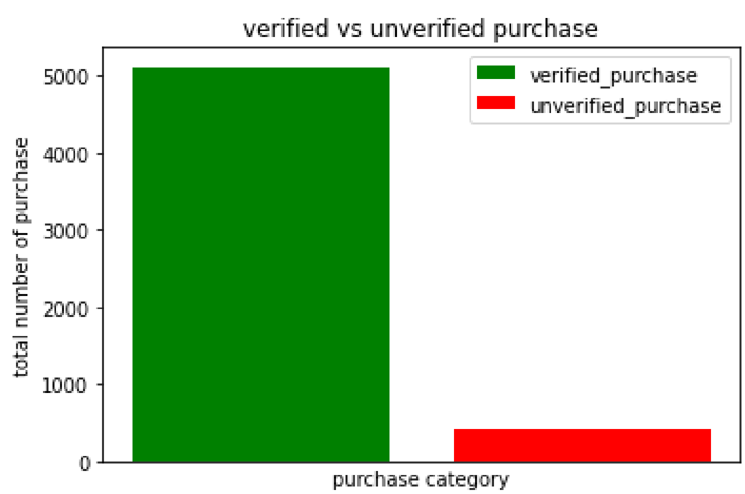
Out[ ]:
```
415
```

Out[ ]:
```
5117
```

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

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

Out[ ]:
```
<matplotlib.legend.Legend at 0x7fc3ff6849b0>
```



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

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

```
Out[ ]: [(97, 'B0091JKY0M'),
         (106, 'BT00CTOUNS'),
         (121, 'BT00DDC7CE'),
         (123, 'B00A48G0D4'),
         (125, 'B0091JKU5Q'),
         (129, 'B007V6EVY2'),
         (139, 'B004KNWWO0'),
         (169, 'B00IX1I3G6'),
         (186, 'BT00DDVMVQ'),
         (412, 'B004LLIKVU')]
```

```
In [ ]: nRatings=len(trainingSet)
        nRatings
```

```
Out[ ]: 5532
```

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

```
Out[ ]: 4.803687635574837
```

```
In [ ]: #total customer headcount is 3511 and products count is 777
```

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

```
Out[ ]: (3511, 777)
```

```
In [ ]: avverified=0
        avunverified=0
        nverified=0
        nunverified=0
        for d in trainingSet:
          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[ ]: (4.810435802227868, 4.720481927710844)
```

In [ ]:
```
'''Average for Verified rating is 4.8104.
Average for unverified rating also is somewhere nearby i.e. 4.7204'''
```

Out[ ]:
```
'Average for Verified rating is 4.8104. \nAverage for unverified ratin
g also is somewhere nearby i.e. 4.7204'
```

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

Out[ ]:
```
4.810435802227868
```

Out[ ]:
```
4.720481927710844
```

Out[ ]:
```
{'customer_id': '43417851',
 'helpful_votes': 0,
 'marketplace': 'US',
 'product_category': 'Gift Card',
 'product_id': 'B00A44A3Y0',
 'product_parent': '578402716',
 'product_title': 'Amazon Gift Card - Print - Happy Birthday (Candle
s)',
 'review_body': 'I trust the gift card was enjoyed. My son-in-law gets
e-books and that makes me believe he enjoyed the gift card.',
 'review_date': '2013-06-07',
 'review_headline': 'You cannot go wrong with a giftcard.',
 'review_id': 'R2YQBK30N9OUCZ',
 'star_rating': 5,
 'total_votes': 0,
 'verified_purchase': True,
 'vine': False,
 'yearint': 2013}
```

Task 2: Classification

Next you will use our knowledge of classification to extract features and make predictions based on them. Here you will be using a Logistic Regression Model, keep this in mind so you know where to get help from.

TODO 1: Define the feature function

This implementation will be based on ***any two*** attributes from your dataset. You will be using these two attributes to predict a third. Hint: Remember the offset!

In [ ]:
```
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)
dataset=[d for d in dataset if 'review_date' in d ]
print('\n')
print(len(dataset))
print(dataset[0])
```

```
148309
{'marketplace': 'US', 'customer_id': '24371595', 'review_id': 'R27ZP1F
1CD0C3Y', 'product_id': 'B004LLIL5A', 'product_parent': '346014806',
'product_title': 'Amazon eGift Card - Celebrate', 'product_category':
'Gift Card', 'star_rating': 5, 'helpful_votes': 0, 'total_votes': 0,
'vine': False, 'verified_purchase': True, 'review_headline': 'Five Sta
rs', 'review_body': 'Great birthday gift for a young adult.', 'review_
date': '2015-08-31'}
```

In [ ]:
```
#FIX THIS
def feature(d):
    feat = [1, d['star_rating'], len(d['review_body'])]
    return feat
```

TODO 2: Fit your model

1. Create your **Feature Vector** based on your feature function defined above.
2. Create your **Label Vector** based on the "verified purchase" column of your training set.
3. Define your model as a **Logistic Regression** model.
4. Fit your model.

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

Out[ ]:
```
[True, True, True, True, True, True, True, True, False, True]
```

In [ ]:
```
#YOUR CODE HERE
model=linear_model.LogisticRegression(max_iter=1000)
model.fit(x,y)
```

Out[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verb
ose=0,
                   warm_start=False)
```

TODO 3: Compute Accuracy of Your Model

1. Make **Predictions** based on your model.
2. Compute the **Accuracy** of your model.

In [ ]:
```
#YOUR CODE HERE
predictions=model.predict(x)
correct=predictions==y
accuracy=sum(correct)/len(correct)
print("accuracy="+str(accuracy))
```

```
accuracy=0.9105381332218544
```

In [ ]:
```
TP=sum([(p and l) for (p,l) in zip(predictions,y)])
FP= sum([(p and not l)for (p,l) in zip(predictions,y)])
TN=sum([(not p and not l) for (p,l) in zip(predictions,y)])
FN=sum([(not p and l)  for (p,l) in zip(predictions,y)])
```

In [ ]:
```
print("TP="+str(TP))
print("FP="+str(FP))
print("TN="+str(TN))
print("FN="+str(FN))
```

```
TP=134915
FP=12895
TN=126
FN=373
```

In [ ]:
```
accuracy=(TP+TN)/(TP+FP+TN+FN)
accuracy
```

Out[ ]:
```
0.9105381332218544
```

In [ ]:
```
TPR=TP/(TP+FN)
TNR=TN/(TN+FP)
BER=1-(1/2*(TPR+TNR))
BER
```

Out[ ]:
```
0.4965402025256064
```

In [ ]:
```
precision=TP/(TP+FP)
precision
```
Out[ ]:
```
0.9127596238414181
```
In [ ]:
```
recall=TP/(TP+TN)
recall
```
Out[ ]:
```
0.9990669500373961
```
In [ ]:
```
F1=2*(precision*recall)/(precision+recall)
F1
```
Out[ ]:
```
0.953965161869677
```
In [ ]:
```
confidences=model.decision_function(x)
confidences
```
Out[ ]:
```
array([2.66074512, 2.47216494, 2.76360704, ..., 1.01873777, 0.9063779
5,
       2.69503243])
```
In [ ]:
```
confidencesandlabels=list(zip(confidences,y))
confidencesandlabels[0:10]
```
Out[ ]:
```
[(2.660745123723817, True),
 (2.472164938651914, True),
 (2.7636070428539457, True),
 (2.3378729886764678, True),
 (2.2978711312369735, True),
 (2.3407302642078607, True),
 (1.7435596781468345, True),
 (2.5407395514053333, True),
 (2.443592183337989, False),
 (2.5235958982169784, True)]
```
In [ ]:
```
labelsrankedbyconfidence=[z[1] for z in confidencesandlabels]
labelsrankedbyconfidence[0:10]
```
Out[ ]:
```
[True, True, True, True, True, True, True, True, False, True]
```
In [ ]:
```
def precisionatk(k,y_sorted):
  return sum(y_sorted[:k])/k
def recallatk(k,y_sorted):
  return sum(y_sorted[:k])/sum(y_sorted)
print(precisionatk(50,labelsrankedbyconfidence))
print(precisionatk(1000,labelsrankedbyconfidence))
print(precisionatk(10000,labelsrankedbyconfidence))
```
```
0.88
0.912
0.9144
```

Task 3: Regression

In this section you will start by working though two examples of altering features to further differentiate. Then you will work through how to evaluate a Regularaized model.

In [ ]:
```
#CHANGE PATH
path = "amazon_reviews_us_Gift_Card_v1_00.tsv.gz"

#GIVEN
f = gzip.open(path, 'rt', encoding="utf8")
header = f.readline()
header = header.strip().split('\t')
reg_dataset = []
for line in f:
    fields = line.strip().split('\t')
    d = dict(zip(header, fields))
    d['star_rating'] = int(d['star_rating'])
    reg_dataset.append(d)
```

TODO 1: Unique Words in a Sample Set

We are going to work with a new dataset here, as such we are going to take a smaller portion of the set and call it a Sample Set. This is because stemming on the normal training set will take a very long time. (Feel free to change sampleSet -> reg_dataset if you would like to see the difference for yourself)

1. Count the number of unique words found within the 'review body' portion of the sample set defined below, making sure to **Ignore Punctuation and Capitalization**.
2. Count the number of unique words found within the 'review body' portion of the sample set defined below, this time with use of **Stemming, Ignoring Puctuation, *and* Capitalization**.

In [ ]:
```
#GIVEN for 1.
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

#GIVEN for 2.
wordCountStem = defaultdict(int)
stemmer = PorterStemmer() #use stemmer.stem(stuff)

#SampleSet and y vector given
sampleSet = reg_dataset[:2*len(reg_dataset)//10]
y_reg = [d['star_rating'] for d in sampleSet]
```

TODO 2: Evaluating Classifiers

1. Given the feature function and your counts vector, **Define** your X_reg vector. (This being the X vector, simply labeled for the Regression model)
2. **Fit** your model using a **Ridge Model** with (alpha = 1.0, fit_intercept = True).
3. Using your model, **Make your Predictions**.
4. Find the **MSE** between your predictions and your y_reg vector.

In [ ]:

```
#GIVEN FUNCTIONS
def feature_reg(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_body'].lower() if not c in punctuation])
    r=[stemmer.stem(w) for w in r.split()]
    r="".join([c for c in r])
    for w in r.split():
        if w in wordSet:
            feat[wordId[w]] += 1
    return feat

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

In [ ]:
```
#GIVEN COUNTS AND SETS
for d in sampleSet:
  r="".join([c for c in d['review_body'].lower() if c not in punctuat
ion])
  for w in r.split():
    wordCount[w]+=1
print("len(wordCount) is ")
print(len(wordCount))
for d in sampleSet:
  r="".join([c for c in d['review_body'].lower() if c not in punctuat
ion])
  r=[stemmer.stem(w) for w in r.split()]
  r="".join([c for c in r])
  for w in r.split():
    wordCountStem[w]+=1
print("len(wordCountStem) is ")
print(len(wordCountStem))

counts = [(wordCountStem[w], w) for w in wordCountStem]
counts.sort()
counts.reverse()
#Note: increasing the size of the dictionary may require a lot of mem
ory
words = [x[1] for x in counts[:100]]
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
print("len(wordSet) is ")
print(len(wordSet))
print("len(words) is ")
print(len(words))
```

```
len(wordCount) is
10765
len(wordCountStem) is
19546
len(wordSet) is
100
len(words) is
100
```

In [ ]:
```
#YOUR CODE HERE
import random
import numpy
random.shuffle(sampleSet)
x_reg=[feature(d) for d in sampleSet]
print(len(x_reg))
print(len(y_reg))
print(x_reg[0])
```

```
29817
29817
[1, 5, 90]
```

In [ ]:
```
N=len(x_reg)
x_train=x_reg[:4*N//5]
x_valid=x_reg[4*N//5:9*N//10]
x_test=x_reg[9*N//10:]
y_train=y_reg[:4*N//5]
y_valid=y_reg[4*N//5:9*N//10]
y_test=y_reg[9*N//10:]
```

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

```
29817
23853
2982
2982
```

In [ ]:
```
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 [ ]:
```
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[ ]:
```
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=0.6833073526505993
0.43995999639773214
```

Out[ ]:
```
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=0.6833073684689341
0.43996016817135764
```

Out[ ]:
```
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=0.6833089452805271
0.43996293839331796
```

Out[ ]:
```
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=0.6834617384449623
0.44009229789893606
```

Out[ ]:
```
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=0.6948844046207753
0.4487251309420954
```

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

```
Ridge(alpha=0.01, copy_X=True, fit_intercept=False, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
0.43995999639773214
```

In [ ]:
```
mseTest=MSE(bestModel,x_test,y_test)
print("testerror"+str(mseTest))
```

```
testerror0.4048214914604578
```

Task 4: Recommendation Systems

For your final task, you will use your knowledge of simple similarity-based recommender systems to make calculate the most similar items.

The next cell contains some starter code that you will need for your tasks in this section. Notice you should be back to using your **trainingSet**.

In [ ]:
```
#GIVEN
attribute_1 = defaultdict(set)
attribute_2 = defaultdict(set)
len(dataset)
```

Out[ ]: 148309

TODO 1: Fill your Dictionaries

1. For each entry in your training set, fill your default dictionaries (defined above).

In [ ]:
```
#YOUR CODE HERE
for d in trainingSet:
  user,item=d['customer_id'],d['product_id']
  attribute_1[item].add(user)
  attribute_2[user].add(item)
```

In [ ]:
```
#GIVEN
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom

def mostSimilar(n, m): #n is the entry index
    similarities = []  #m is the number of entries
    users = attribute_1[n]
    for i2 in attribute_1:
        if i2 == n: continue
        sim = Jaccard(users, attribute_1[n])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:m]
```

TODO 1: Fill your Dictionaries

1. Calculate the **10** most similar entries to the **first** entry in your dataset, using the functions defined above.

```
In [ ]: #YOUR CODE HERE
        query=trainingSet[2]['product_id']
        query
```

```
Out[ ]: 'B007RFEL42'
```

```
In [ ]: mostSimilar(query,10)
```

```
Out[ ]: [(1.0, 'BT00DDVMVQ'),
         (1.0, 'BT00DDC88W'),
         (1.0, 'BT00DDC7CE'),
         (1.0, 'BT00DDC7C4'),
         (1.0, 'BT00DDC7BK'),
         (1.0, 'BT00DDBSA6'),
         (1.0, 'BT00DC6QU4'),
         (1.0, 'BT00CTPCX0'),
         (1.0, 'BT00CTPCO4'),
         (1.0, 'BT00CTPBMM')]
```

```
In [ ]: #useful data structures
        userspteritem=defaultdict(set)
        itemsperuser=defaultdict(set)
        itemnames={}
        dataset=dataset1
        len(dataset1)
        #len(dataset)
```

```
Out[ ]: 6915
```

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

Out[ ]:
```
[(0.05555555555555555, 'B005ESMGZU'),
 (0.05333333333333334, 'B0080IR4MQ'),
 (0.041666666666666664, 'B005ESMJ02'),
 (0.04, 'B008EN462I'),
 (0.02702702702702703, 'B0091JKFG0'),
 (0.026490066225165563, 'B0091JKY0M'),
 (0.024691358024691357, 'B005ESMF5G'),
 (0.023255813953488372, 'B0091JKZO2'),
 (0.023255813953488372, 'B005DHN642'),
 (0.022727272727272728, 'B0083V8XIE')]
```

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

Out[ ]:
```
4.806073752711497
```

In [ ]:
```python
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(userseperitem[item],userseperitem[i2
]))
  if sum(similiarities)>0:
    weightedratings=[(x*y) for x,y in zip(ratings,similiarities)]
    return sum(weightedratings)/sum(similiarities)
  else:
    return ratingmean
```

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

Out[ ]:
```
3.992438563327032
```

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

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

In [ ]:
```python
'''here MSE doing worse than in case of always predicting the
mean which is 0.4597 compared to 0.9128.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[ ]:
```
'here MSE doing worse than in case of always predicting the \nmean whi
ch is 0.4597 compared to 0.9128.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 [ ]:
```python
labels=[d['star_rating']for d in dataset]
MSE(alwayspredictmean,labels)
```

Out[ ]:
```
0.45971726715627703
```

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

Out[ ]:
```
0.9128137351578797
```

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

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

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

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

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

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

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

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

Out[ ]:
```
0.45971726715627703
```

In [ ]:
```
'''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[ ]:
```
"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= 0.45971726715627703
MSE= 0.4399841504493804
MSE= 0.3369423548052294
MSE= 0.3498790806547602
MSE= 0.3050958666537619
MSE= 0.3217013229920161
MSE= 0.32167350310743503
MSE= 0.3214619355418692
MSE= 0.3213795546170475
MSE= 0.32147743253813743
MSE= 0.32161703857439355
MSE= 0.32166921365118356
MSE= 0.32167877042426246
MSE= 0.321692112062229
MSE= 0.3217010442164441
MSE= 0.3216830941923795
MSE= 0.3217319774762689
MSE= 0.3216891356009209
```

Out[ ]:
```
(array([ 4.80599347, -0.06863377,  0.08123547, ...,  0.        ,
         0.        ,  0.        ]),
 0.3820176685706553,
 {'funcalls': 18,
  'grad': array([ 2.64386039e-06,  3.38048881e-08, -6.10842455e-07,
...,
          0.00000000e+00,  0.00000000e+00,  0.00000000e+00]),
  'nit': 15,
  'task': b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'warnflag': 0})
```

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

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

Finished!

Congratulations! You are now ready to submit your work. Once you have submitted make sure to get started on your peer reviews!