Project for the Course: Meaningful Predictive Modeling.

Description of the Dataset:

The dataset that will be analysed for this project has been downloaded from the Kaggle website. The dataset - 'Flipcart Products Review Dataset' Description: The dataset contains information such as products name, price, review, rating and summary for 104 types of different products on flipcart.com. It has 194276 rows and 5 columns. It was chosen for analysis as it has different customer reviews and can be easily used for sentiment analysis.

To download the data and for more description on the datasets go to: https://www.kaggle.com/datasets/mansithummar67/flipkart-product-review-dataset?resource=download (https://www.kaggle.com/datasets/mansithummar67/flipkart-product-review-dataset?resource=download)

Objectives:

1. Whether the product reviews on a product can be used to determine overall sentiment towards the product.
2. Whether the product reviews can be used for pricing decisions. That is, if a product has higher ratings and positive reviews, the price may be increased, if it has lower ratings and negative reviews, the price may be decreased.

In [1]:

```
1
2 import gzip
3 import string
4 import random
5 from nltk.stem.porter import PorterStemmer
6
7 path = r"D:\TRL\OneDrive\CourseraPython\Final_Course3\flipkart_product.csv"
```

The following code can be used to get the data from the csv file BUT: see the note below.

Note: The following code was first used to read the data from the csv file, but it does not work the way that it was intended to because the csv file as well as the Product descriptions in the first column are comma separated. Because of this the product description spills over to the next column instead of the next column entry. Also using any other delimiter does not help with this problem. So I have researched pandas and used them.

f = open(path,'rt', encoding = "ISO-8859-1")

import csv reader = csv.reader(f, delimiter = ',')

header = f.readline() header

header = header.strip().split(',') header

d = next(reader) # there is some problem with the csv file, Prices in all d # entries have garbage before the price value, which I have to try and remove

dataset = [] for line in f: fields = line.strip().split('\t') d = dict(zip(header,fields)) # keys in the header are mapped for field in ['Price', 'Rate']: # to values in each line d[field] = int(d[field])
dataset.append(d)

print("len(d) =",len(dataset)) d

dataset[10] # testing

dataset[189871]

The following also doesn't work:

data_new = [] for lines in range(len(dataset)): data_new.append(lines)

type(dataset)

type(data_new)

In [2]:

```
1 ### using pandas and dataframe to read the csv file as it cannot be
2 # read as dictionary object using a for loop
3
4 import pandas as pd
5 #df = pd.read_csv("D:\TRL\OneDrive\CourseraPython\Final_Course3\flipkart_product.csv")
```

In [3]:

```
1 import chardet
2 with open(path, 'rb') as rawdata:
3     result = chardet.detect(rawdata.read(189872))
```

In [4]:

```
1 result
```

Out[4]:

```
{'encoding': 'Windows-1254',
 'confidence': 0.4569690721335204,
 'language': 'Turkish'}
```

In [ ]:

```
dataset = pd.read_csv(r"D:\TRL\OneDrive\CourseraPython\Final_Course3\flipkart_product.csv", encoding ="ISO-8859-1",dtype = {'ProductN
```

In [6]:

```
dataset
```

Out[6]:

	ProductName	Price	Rate	Review	Summary
0	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	5	Super!	Great cooler.. excellent air flow and for this...
1	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	5	Awesome	Best budget 2 fit cooler. Nice cooling
2	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	3	Fair	The quality is good but the power of air is de...
3	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	1	Useless product	Very bad product it's a only a fan
4	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	3	Fair	Ok ok product
...	...	...	...	...	...
189868	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Good
189869	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Thanks
189870	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Good
189871	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Just wow!	Super
189872	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	4	Worth the money	Good

189873 rows × 5 columns

In [7]:

```
dataset.iloc[1,:]
```

Out[7]:

```
ProductName    Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...
Price                                                       3999
Rate                                                           5
Review                                                   Awesome
Summary                Best budget 2 fit cooler. Nice cooling
Name: 1, dtype: object
```

In [8]:

```
dataset.iloc[1,2]
```

Out[8]:

```
'5'
```

In [9]:

```
from collections import defaultdict
import numpy as np
```

In [10]:

```
# for the sake of analysis, let us randomly generate a column of number of
# helpful reviews, with the number of helpful reviews between 1 to 100.

dataset['Helpful_Reviews'] = np.random.randint(1,100,dataset.shape[0])
```

In [11]:

```
dataset
```

Out[11]:

	ProductName	Price	Rate	Review	Summary	Helpful_Reviews
0	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	5	Super!	Great cooler.. excellent air flow and for this...	61
1	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	5	Awesome	Best budget 2 fit cooler. Nice cooling	49
2	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	3	Fair	The quality is good but the power of air is de...	45
3	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	1	Useless product	Very bad product it's a only a fan	5
4	Candes 12 L Room/Personal Air Cooler?ÿ?ÿ(White...	3999	3	Fair	Ok ok product	57
...	...	...	...	...	...	...
189868	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Good	36
189869	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Thanks	12
189870	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Terrific	Good	51
189871	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	5	Just wow!	Super	22
189872	NIVEA Soft Light Moisturizer for Face, Hand & ...	142	4	Worth the money	Good	64

189873 rows × 6 columns

In [12]:

```
# since the last row is NaN, we delete it
# dataset.drop(189873) # no need to run it the second time
```

In [13]:

```
dataset.iloc[1][1] +  dataset.iloc[2][1], dataset.iloc[1][2] + dataset.iloc[2][2], dataset.iloc[1][5] + dataset.iloc[2][5]

# this shows that the values in the second column and third column are
# strings, and on adding, will be concatenated, the values in the sixth
# column can be added etc
```

Out[13]:

```
('39993999', '53', 94)
```

In [14]:

```
int(dataset.iloc[1][1]) +  int(dataset.iloc[2][1]) # we have to convert to add
```

Out[14]:

```
7998
```

In [17]:

```
indices = []
for i in range(len(dataset)):
    if len(str(dataset.iloc[i][2])) > 2 :
        indices.append(i)
        print(i)
```

```
40725
82709
82720
```

In [18]:

```
indices
dataset.iloc[indices[0]][2], dataset.iloc[indices[1]][2], dataset.iloc[indices[2]][2]
```

Out[18]:

```
('Pigeon Favourite Electric Kettle?ÿ?ÿ(1.5 L, Silver, Black)',
 'Bajaj DX 2 L/W Dry Iron',
 'Nova Plus Amaze NI 10 1100 W Dry Iron?ÿ?ÿ(Grey & Turquoise)')
```

In [19]:

```
# these three cells were deleted for analyses as no number was present
# in the Price and rating fields.
dataset.drop([indices[0],indices[1],indices[2]], inplace = True) # inplace = True updates the dataframe
```

In [20]:

```
dataset.iloc[indices[2]] # this shows that the rows corresponding to these
# indices have been deleted
```

Out[20]:

```
ProductName        Nova Plus Amaze NI 10 1100 W Dry Iron?ÿ?ÿ(Grey...
Price                                                            494
Rate                                                               5
Review                                                      Terrific
Summary            Fantastic look, Fast producing heat, Light wei...
Helpful_Reviews                                                   38
Name: 82723, dtype: object
```

In [21]:

```
len(dataset)
```

Out[21]:

```
189870
```

In [22]:

```
# Our goal is to build a classifier that estimates the sentiment- the star
# rating, based on the occurance of words in the review.
# Note: the column 4 of the dataset has been derived using the Summary.

# For this, we first count the number of unique words in the reviews
wordCount = defaultdict(int)
for i in range(len(dataset)):
    for w in str(dataset.iloc[i][4]).split(): # see code in cells below for how this works
        wordCount[w] += 1

print(len(wordCount))

```

```
82191
```

In [23]:

```
type(dataset['Summary'][2]) ,dataset['Summary'][3]
```

Out[23]:

```
(str, "Very bad product it's a only a fan")
```

In [24]:

```
dataset.iloc[2][4], dataset.iloc[3][4]
```

Out[24]:

```
('The quality is good but the power of air is decent',
 "Very bad product it's a only a fan")
```

In [25]:

```
len(dataset)
```

Out[25]:

```
189870
```

In [26]:

```
dataset['Summary'][5].split()
```

Out[26]:

```
['The',
 'cooler',
 'is',
 'really',
 'fantastic',
 'and',
 'provides',
 'good',
 'air',
 'flow.',
 'Highly',
 'recommended']
```

In [27]:

```
range(len(dataset))
```

Out[27]:

```
range(0, 189870)
```

In [28]:

```
len(wordCount)
```

Out[28]:

```
82191
```

In [29]:

```
# Now let us ignore the capitalization and punctuation
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

for i in range(len(dataset)): #dataset.['Summary'][i] can be used instead of dataset['Summary'][i]
    s = ''.join([c for c in str(dataset.iloc[i][4]).lower() if not c in punctuation])
    for w in s.split():
        wordCount[w] += 1

print(len(wordCount))
# it did reduce by about 30000
```

```
49684
```

In [30]:

```
# Now let us try stemming to see the unique stems of the words in the reviews
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
stemmer = PorterStemmer()

for i in range(len(dataset)):
    s = ''.join([c for c in str(dataset.iloc[i][4]).lower() if not c in punctuation])
    for w in s.split():
        w = stemmer.stem(w)
        wordCount[w] += 1

print(len(wordCount))
# it further reduced
```

```
43318
```

In [31]:

```
# Now let us extract the most common words
count_unique = [(wordCount[w],w) for w in wordCount]
count_unique.sort()
count_unique.reverse()
my_words = [w[1] for w in count_unique[:1000]]
```

In [32]:

```
my_words
```

Out[32]:

```
['good',
 'product',
 'is',
 'veri',
 'it',
 'nice',
 'and',
 'the',
 'for',
 'qualiti',
 'thi',
 'i',
 'to',
 'not',
 'in',
 'of',
 'a',
 'but'.
```

In [33]:

```
my_words_id = dict(zip(my_words,range(len(my_words))))
```

In [34]:

```
my_words_id['best']  # example
```

Out[34]:

```
18
```

In [35]:

```
my_words_set = set(my_words)
```

In [36]:

```
# Now these words correspond to feature indices
# i.e. each review can be mapped to a 1000 dimensional vector

def feature(datarow):
    feat = [0]*len(my_words)
    s = ''.join([c for c in str(dataset.iloc[datarow][4]).lower() if not c in punctuation])
    for w in s.split():
        if w in my_words:
            feat[my_words_id[w]] += 1
    feat.append(1) # this is for the offset feature
    return feat
```

In [37]:

```
# Now, for each review, we can calculate rating as follows:
# rating = alpha + (sum over word) count(word) * theta(word)
# theta(word) is the amount by which rating is affected for the 'word'
# in the review.

# Now to fit this model, we first shuffle the dataset
# instead of the function shuffle, we use the fucntion sample for dataframes
dataset = dataset.sample(frac=1)
dataset
```

Out[37]:

	ProductName	Price	Rate	Review	Summary	Helpful_Reviews
89353	MILTON Thermosteel Flip Lid 500 ml FlaskÂ Â (P...	648	5	Excellent	Its very good	99
50728	Saral Home Jute Door MatÂ Â (Green, Medium, Pa...	484	5	Great product	Good	83
185636	Butterfly Rhino Plus Wet Grinder?ÿ?ÿ(Grey)	5299	4	Very Good	Best in the price	72
56843	Men Black Sandal	299	5	Nan	super	75
86391	HARIOM ALL IN ONE Solid Wood Study TableÂ Â (F...	549	3	Good	Table stand is not fit	18
...	...	...	...	...	...	...
61576	Google Nest Mini (2nd Gen) with Google Assista...	3499	1	Unsatisfactory	Very bad product	89
13273	Hindware Nevio Plus 60 Auto Clean Wall Mounte...	13999	5	Brilliant	Great product, timely delivered. Thanks Flipka...	49
7144	Cosito 144 TC Cotton Double Floral Flat Bedshe...	339	1	Worst experience ever!	Not good because it's quilaty is worst not goo...	99
181132	BAJAJ 3 L Instant Water Geyser (Ivora/ flora, ...	2599	5	Fabulous!	Good product	84
20062	SportSoul Cotton Gym & Athletic Abdomen Suppor...	249	5	Excellent	Nice product and comfortable also	76

189870 rows × 6 columns

In [38]:

```
dataset.iloc[0] # see how it has changed now
```

Out[38]:

```
ProductName        MILTON Thermosteel Flip Lid 500 ml FlaskÂ Â (P...
Price                                                           648
Rate                                                              5
Review                                                    Excellent
Summary                                                Its very good
Helpful_Reviews                                                  99
Name: 89353, dtype: object
```

In [39]:

```
X = [feature(i) for i in range(len(dataset))] # the feature matrix
X[1]
```

Out[39]:

```
[1,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0.
```

In [40]:

```
dataset['Summary'][1] # this is the 2nd summary in the original dataset
```

Out[40]:

```
'Best budget 2 fit cooler. Nice cooling'
```

In [41]:

```
dataset.iloc[1][4] # this is the summary in the shuffled dataset.
# Note that the 1s in X[1] correspond to the review below.
```

Out[41]:

```
'Good'
```

In [42]:

```
# dimensions of the matrix X:
len(X), len(X[0]), len(dataset)
```

Out[42]:

```
(189870, 1001, 189870)
```

In [43]:

```

```

In [44]:

```
dataset.iloc[98411]
```

Out[44]:

```
ProductName        cello Pack of 18 Opalware Cello Dazzle Lush Fi...
Price                                                           1299
Rate                                                               3
Review                                                          Good
Summary            quality is good but i found one broken bowl. p...
Helpful_Reviews                                                   40
Name: 169875, dtype: object
```

In [45]:

```
y = [int(dataset.iloc[i][2]) for i in range(len(dataset))] #extracting the labels for each datapoint
```

In [46]:

```
type(y), type(y[1])
```

Out[46]:

```
(list, int)
```

In [47]:

```
type(dataset['Price'][1])
```

Out[47]:

```
str
```

In [48]:

```
import numpy
theta, residuals, rank, s = numpy.linalg.lstsq(X,y, rcond= -1)
```

In [49]:

```
theta
```

Out[49]:

```
array([ 0.31935216,  0.07326229, -0.01697148, ...,  0.04540193,
        0.        ,  4.04208313])
```

In [50]:

```
len(theta)
```

Out[50]:

```
1001
```

In [52]:

```
# words that have the most positive or negative sentiment:
wordSentiments = list(zip(theta, my_words + ['offset']))
wordSentiments.sort()
```

In [54]:

```
vec = my_words + ['offset']
len(vec), vec[1000]
```

Out[54]:

```
(1001, 'offset')
```

In [55]:

```
wordSentiments[1:]
```

Out[55]:

```
[(-158254462064.39267, 'littl'),
 (-119127660731.73515, 'beauti'),
 (-64329410242.87878, 'experi'),
 (-29062485098.33034, 'deliv'),
 (-19866341340.955112, 'receiv'),
 (-14808177446.282583, 'materi'),
 (-2367638396.4063454, 'properli'),
 (-1915817829.1457095, 'fridg'),
 (-97958114.31251046, 'cycl'),
 (-82547846.8086258, 'everyth'),
 (-40678272.107992135, 'fantast'),
 (-2067402.7352560484, 'tabl'),
 (-1797726.353305803, 'googl'),
 (-59607.25697928499, 'usag'),
 (-16412.507813690943, 'assembl'),
 (-15819.51320029821, 'sinc'),
 (-8372.694623059942, 'easili'),
 (-6442.748911438002, 'chang')]
```

In [56]:

```
wordSentiments[-10:]
```

Out[56]:

```
[(248868.8922112999, 'compani'),
 (581033.7833893094, 'condit'),
 (977221.1691484333, 'heavi'),
 (136463205.74634764, 'devic'),
 (601973899.2965547, 'becaus'),
 (1915817829.4525018, 'nois'),
 (2237730774.379421, 'batteri'),
 (76453055425.66623, 'servic'),
 (82040375053.65384, 'ani'),
 (116879229366.5479, 'realli')]
```

In [57]:

```
from sklearn import linear_model
```

In [58]:

```
help(linear_model.Ridge)
```

```
Help on class Ridge in module sklearn.linear_model._ridge:

class Ridge(sklearn.base.MultiOutputMixin, sklearn.base.RegressorMixin, _BaseRidge)
 |  Ridge(alpha=1.0, *, fit_intercept=True, normalize='deprecated', copy_X=True, max_iter=None, tol=0.001, solver='auto',
positive=False, random_state=None)
 |
 |  Linear least squares with l2 regularization.
 |
 |  Minimizes the objective function::
 |
 |  ||y - Xw||^2_2 + alpha * ||w||^2_2
 |
 |  This model solves a regression model where the loss function is
 |  the linear least squares function and regularization is given by
 |  the l2-norm. Also known as Ridge Regression or Tikhonov regularization.
 |  This estimator has built-in support for multi-variate regression
 |  (i.e., when y is a 2d-array of shape (n_samples, n_targets)).
 |
 |  Read more in the :ref:`User Guide <ridge_regression>`.
 '
```

In [59]:

```
# let us try ridge regression now with regularization strength 3

model = linear_model.Ridge(3.0, fit_intercept = False)
model.fit(X,y)
```

Out[59]:

```
Ridge(alpha=3.0, fit_intercept=False)
```

In [60]:

```
theta = model.coef_
```

In [62]:

```
wordSentimentsNew = list(zip(theta, my_words + ['offset']))
wordSentimentsNew.sort()
```

In [63]:

```
wordSentimentsNew[:10]
```

Out[63]:

```
[(-1.8727821610012447, 'poor'),
 (-1.801633747380397, 'wrost'),
 (-1.6433894486160834, 'worst'),
 (-1.6345718876250876, 'bad'),
 (-1.4576261938967328, 'useless'),
 (-1.4266942559614328, 'wast'),
 (-1.3527711098543473, 'third'),
 (-1.335546347037573, 'bed'),
 (-1.0944238213126245, 'broken'),
 (-1.0680260658979142, 'hate')]
```

In [64]:

```
# MSE and R2 statistic

RRprediction = model.predict(X) # prediction using ridge regression
```

In [66]:

```
RRprediction, len(RRprediction)
```

Out[66]:

```
(array([4.36117795, 4.36117795, 4.51824222, ..., 0.90841966, 4.43444389,
        4.61213638]),
 189870)
```

In [65]:

```
diff = [(a-b)**2 for (a,b) in zip(RRprediction,y)]
```

In [68]:

```
MSE = sum(diff)/len(diff)
print("MSE = " + str(MSE))
```

```
MSE = 0.9225290739655049
```

In [70]:

```
FVU = MSE/ numpy.var(y)
FVU
```

Out[70]:

```
0.5418077253507845
```

In [73]:

```
R2 = 1-FVU
print("R2 = " + str(R2))
```

```
R2 = 0.4581922746492155
```

In [78]:

```
# let us look at this as a classification problem
yCategorical = [(rating > 3) for rating in y]
```

In [81]:

```
model = linear_model.LogisticRegression(solver = 'lbfgs', max_iter = 500)
model.fit(X,yCategorical)
# had to increase the number of iterations as the solver did not converge in 100 iterations
```

Out[81]:

```
LogisticRegression(max_iter=500)
```

In [84]:

```
# Computation of Accuracy of the classifier
predictions = model.predict(X)
correct = predictions == yCategorical
```

In [86]:

```
accuracy = sum(correct)/ len(correct)
accuracy
```

Out[86]:

```
0.8995839258439985
```

In [87]:

```
#  True Positive: Prediction is True and label is True
TP = sum([(p and l) for (p,l) in zip(predictions, yCategorical)]) # 1 and 1 = 1 etc
# False Positive: the prediction is True when the label is False.
FP = sum([(p and not l) for (p,l) in zip(predictions, yCategorical)]) # 1 and not 1 = 0
# True negative: prediction is False when actual label is False.
TN = sum([(not p and not l) for (p,l) in zip(predictions, yCategorical)]) # not 1 and not 1 = 0
# False Negative: prediction is False when the the actual label is True
FN = sum([(not p and l) for (p,l) in zip(predictions, yCategorical)])  # not 1 and 1 = 0

```

In [88]:

```
TPR = TP/(TP+FN)
TNR = TN/(TP+FN)
```

In [90]:

```
BER = 1-1/2*(TPR+TNR)
BER # Balanced error rate
```

Out[90]:

```
0.42431310162590663
```

In [92]:

```
precision = TP/ (TP+FP)
precision
```

Out[92]:

```
0.9021463232549459
```

In [93]:

```
recall = TP / (TP+FN)
recall
```

Out[93]:

```
0.9775055949524092
```

In [94]:

```
F1 = 2*((precision*recall)/ (precision+recall))
```

In [95]:

```
F1
```

Out[95]:

```
0.9383153017910758
```

In [96]:

```
# Confidence scores per sample:
confidences = model.decision_function(X)
```

In [97]:

```
confidences
```

Out[97]:

```
array([ 2.11753817,  2.11753817,  3.27800545, ..., -6.77942346,
        2.42748215,  3.40510561])
```

In [98]:

```
len(confidences)
```

Out[98]:

```
189870
```

In [100]:

```
confidenceLabels = list(zip(confidences,yCategorical))
```

In [101]:

```
confidenceLabels
```

Out[101]:

```
[(2.1175381722499567, True),
 (2.1175381722499567, True),
 (3.278005448773111, True),
 (3.112194695039198, True),
 (-1.9165031678970417, False),
 (7.5066088812710685, True),
 (3.324861824331816, True),
 (4.479214485763893, True),
 (2.1175381722499567, True),
 (-2.955657746575442, False),
 (2.1175381722499567, False),
 (2.847160086211357, True),
 (2.428779031464267, True),
 (2.01084309481325, True),
 (7.441380168165194, True),
 (2.1175381722499567, True),
 (-0.45650721858336285, True),
 (2.524534701165583, True),
```

In [102]:

```
confidenceLabels.sort(reverse = True)
```

In [103]:

```
# labels ranked by confidence
labelsRanked = [z[1] for z in confidenceLabels]
```

In [118]:

```
def precisionAtK(K,sortedList):
    return sum(sortedList[:K])/K
```

In [119]:

```
def recallAtK(K,sortedList):
    return sum(sortedList[:K]) / sum(sortedList)
```

In [120]:

```
precisionAtK(100,labelsRanked)
```

Out[120]:

```
0.98
```

In [121]:

```
precisionAtK(1000,labelsRanked)
```

Out[121]:

```
0.992
```

In [122]:

```
precisionAtK(10000,labelsRanked)
```

Out[122]:

```
0.9802
```

In [123]:

```
precisionAtK(100000,labelsRanked)
```

Out[123]:

```
0.95238
```

In [124]:

```
recallAtK(100,labelsRanked)
```

Out[124]:

```
0.0006606088386766252
```

In [125]:

```
recallAtK(1000,labelsRanked)
```

Out[125]:

```
0.006686979264971553
```

In [126]:

```
recallAtK(10000,labelsRanked)
```

Out[126]:

```
0.06607436568069674
```

In [127]:

```
recallAtK(100000,labelsRanked)
```

Out[127]:

```
0.6419904548763717
```

In [128]:

```
# Validation pipeline:
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 [129]:

```
len(X), len(X_train), len(X_valid), len(X_test)
```

Out[129]:

```
(189870, 94935, 47467, 47468)
```

In [130]:

```
# To calculate MSE for a particular model:
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 [132]:

```
# Train the model for different regularization parameters

bestModel = None
bestMSE = None

for lamb in [0.01, 0.1, 1, 10, 100]:
    model = linear_model.Ridge(lamb, fit_intercept = False) # fit a model for each lambda value
    model.fit(X_train, y_train)

    mseTrain = MSE(model, X_train, y_train)
    mseValid = MSE(model, X_valid, y_valid)

    print("lambda = " + str(lamb) +", training/validation error = " +
         str(mseTrain) + '/' + str(mseValid)) # report the training and validation error

    if not bestModel or mseValid < bestMSE:
        bestModel = model
        bestMSE = mseValid
```

```
lambda = 0.01, training/validation error = 0.9192329263206979/0.9462662313693249
lambda = 0.1, training/validation error = 0.9192425105472739/0.9459396655838058
lambda = 1, training/validation error = 0.9195268042641338/0.9443912918642349
lambda = 10, training/validation error = 0.9208521139900501/0.9414169466619998
lambda = 100, training/validation error = 0.9319774227720088/0.9418901298269216
```

In [134]:

```
# best Model with least error
mseTest = MSE(bestModel, X_test, y_test)
print("test error = " + str(mseTest))
bestModel
```

```
test error = 0.9323989084655248
```

Out[134]:

```
Ridge(alpha=10, fit_intercept=False)
```

In [135]:

```
bestMSE
```

Out[135]:

```
0.9414169466619998
```

In [ ]:

```

```