1. Number of reviews for each style of beer:

CodeStub:

```
BeerStyle = [d['beer/style'] for d in data]
for b in Counter(BeerStyle):
        print(b)
        BeerStyleTaste = [d['review/taste'] for d in data if d['beer/style']
== b]
        print(np.average(BeerStyleTaste))
print(Counter(BeerStyle))
```

Answer:

Counter({'American Double / Imperial Stout': 5964, 'American IPA': 4113, 'American Double / Imperial IPA': 3886, 'Scotch Ale / Wee Heavy': 2776, 'Russian Imperial Stout': 2695, 'American Pale Ale (APA)': 2288, 'American Porter': 2230, 'Rauchbier': 1938, 'Rye Beer': 1798, 'Czech Pilsener': 1501, 'Fruit / Vegetable Beer': 1355, 'English Pale Ale': 1324, 'Old Ale': 1052, 'Doppelbock': 873, 'American Barleywine': 825, 'Euro Pale Lager': 701, 'Extra Special / Strong Bitter (ESB)': 667, 'American Amber / Red Ale': 665, 'Munich Helles Lager': 650, 'Belgian Strong Pale Ale': 632, 'Hefeweizen': 618, 'American Stout': 591, 'German Pilsener': 586, 'Pumpkin Ale': 560, 'MÃ¤rzen / Oktoberfest': 557, 'Baltic Porter': 514, 'Light Lager': 503, 'English Brown Ale': 495, 'Wheatwine': 455, 'English Porter': 367, 'American Blonde Ale': 357, 'Euro Strong Lager': 329, 'American Brown Ale': 314, 'English Bitter': 267, 'Winter Warmer': 259, 'Tripel': 257, 'American Adjunct Lager': 242, 'Maibock / Helles Bock': 225, 'English India Pale Ale (IPA)': 175, 'Belgian Dark Ale': 175, 'American Strong Ale': 166, 'Altbier': 165, 'Dubbel': 165, 'English Strong Ale': 164, 'Witbier': 162, 'American Pale Wheat Ale': 154, 'Bock': 148, 'Belgian Strong Dark Ale': 146, 'Euro Dark Lager': 144, 'Belgian Pale Ale': 144, 'Munich Dunkel Lager': 141, 'Saison / Farmhouse Ale': 141, 'American Black Ale': 138, 'English Stout': 136, 'English Barleywine': 133, 'Belgian IPA': 128, 'American Pale Lager': 123, 'Black & Tan': 122, 'Quadrupel (Quad)': 119, 'Oatmeal Stout': 102, 'Irish Dry Stout': 101, 'American Wild Ale': 98, 'KÃ¶lsch': 94, 'American Malt Liquor': 90, 'Irish Red Ale': 83, 'Scottish Ale': 78, 'Herbed / Spiced Beer': 73, 'Milk / Sweet Stout': 69, 'Cream Ale': 69, 'Scottish Gruit / Ancient Herbed Ale': 65, 'Smoked Beer': 61, 'Dunkelweizen': 61, 'Foreign / Export Stout': 55, 'Schwarzbier': 53, 'American Amber / Red Lager': 42, 'Vienna Lager': 33, 'Dortmunder / Export Lager': 31, 'Braggot': 26, 'Keller Bier / Zwickel Bier': 23, 'English Dark Mild Ale': 21, 'English Pale Mild Ale': 21, 'American Double / Imperial Pilsner': 14, 'American Dark Wheat Ale': 14, 'Flanders Oud Bruin': 13, 'Weizenbock': 13, 'California Common / Steam Beer': 11, 'Chile Beer': 11, 'Lambic - Unblended': 10, 'Berliner Weissbier': 10, 'Eisbock': 8, 'Low Alcohol Beer': 7, 'BiÃ¨re de Garde': 7, 'Kristalweizen': 7, 'Lambic - Fruit': 6, 'Flanders Red Ale': 2})

Average review/taste for each beer/style:

Hefeweizen
3.63511326861
English Strong Ale
3.75609756098

Foreign / Export Stout
3.25454545455
German Pilsener
3.66723549488
American Double / Imperial IPA
4.03332475553
Herbed / Spiced Beer
3.44520547945
Oatmeal Stout
3.77450980392
American Pale Lager
3.21544715447
Rauchbier
4.06785345717
American Pale Ale (APA)
3.64969405594
American Porter
4.08183856502
Belgian Strong Dark Ale
3.69520547945
Russian Imperial Stout
4.30037105751
American Amber / Red Ale
3.51353383459
American Strong Ale
3.56927710843
MÃ¤rzen / Oktoberfest
3.5933572711
American Adjunct Lager
2.94834710744
American Blonde Ale
3.25490196078
American IPA
4.00085096037
Fruit / Vegetable Beer
3.60774907749
English Bitter
3.53745318352
English Porter
3.70708446866
Irish Dry Stout
3.62376237624
American Barleywine
4.06424242424 ……And so on.

2. Theta values: [3.91520474, 0.08564622]

```
def featureAddition(datum):
    feat = [1]
    if(datum['beer/style'] == 'American IPA'):
        feat.append(1)
    else:
        feat.append(0)
    return feat

X = [featureAddition(d) for d in data]
Y = [d['review/taste'] for d in data]

theta, residuals, rank, s = np.linalg.lstsq(X, Y)
print(theta)
```

Interpretation:

θ_0 is weight assigned to feature x_1=1 which is present for each X_i. θ_1 is the weight associated with the feature x_2 which is added only if the beer is American IPA. What we can infer from this is that the average review/taste for American IPA beer is higher than for the non American IPA beer, hence the positive θ_1.

3. **Theta**: [3.90435639 0.05606027]

Mse Training Error: [0.5581072865586774]

Mse Test Error: [0.4684100509666265]

Code stub:

```
Xtrain, Xtest = np.array_split(X, 2)
Ytrain, Ytest = np.array_split(Y, 2)

theta = np.linalg.lstsq(Xtrain, Ytrain)[0]
print(theta)

def MSE(theta, X, y):
  theta = np.matrix(theta).T
  X = np.matrix(X)
  y = np.matrix(y).T
  diff = X*theta - y
  diffSq = diff.T*diff
  diffSqReg = diffSq / len(X)
  print("offset =", diffSqReg.flatten().tolist())
  return diffSqReg.flatten().tolist()[0]

mseTrain = MSE(theta, Xtrain, Ytrain)
print("Mse Training Error: ",mseTrain)
mseTest = MSE(theta, Xtest, Ytest)
print("Mse Test Error: ", mseTest)
```

4. **Theta values**:

[3.60681818 -0.0058248   0.12193999 -0.35681818 -0.62765152 0.33596789

-0.38622995 0.1527972 -0.58598485 0.44318182 0.02739474 0.3305986

0.12651515 0.69564175 -0.11634199 -0.1798951 -0.22045455 -0.73802386

-0.45410882 0.35359848 0.11320382 -0.02450111 0.09815076 0.22651515

0.45638407 0.8416167 -0.41285266 0.20312334 -0.10681818 -0.83277972

-1.23390152 -0.92019846 -0.74318182 -0.13106061 -0.18884943 0.10049889

0.26709486 0.3763751 -0.17824675 0.71136364 0.12395105 -0.50681818

0.26818182 -0.09003966 0.20984848 0.19621212 0.03420746 0.76540404

0.14466111 0.11433566 0.48544372 -0.31931818 0.46842454 0.16385851

0.05895722 0.26761104 -0.4664673 0.25681818 0.13786267 0.01699134

-0.27061129 -0.63806818 -0.10223103 -0.2937747 0.3449362 0.33580477

-0.65227273 0.58195733 0.38356643 0.01818182 -1.00681818 -0.20681818

0.65508658 0.60472028 0.39318182]

Mse Training Error: [0.3678402770900983]

Mse Test Error: [0.43366951042007407]

Code:

```
FamousBeerStyles = []
BeerStyleReviewCount = Counter(BeerStyle)
for key, value in BeerStyleReviewCount.items():
    if value>=50:
        FamousBeerStyles.append(key)

print(FamousBeerStyles[1])
def featureAddition1(datum):
    feat = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
    beerStyle = datum['beer/style']
    if(beerStyle in FamousBeerStyles):
        index = FamousBeerStyles.index(beerStyle)
        feat[index+1] = 1
    return feat

trainData, testData = np.array_split(data, 2)
XtrainMulti = [featureAddition1(d) for d in trainData]
XtestMulti = [featureAddition1(d) for d  in testData]

theta = np.linalg.lstsq(XtrainMulti, Ytrain)[0]
print(theta)

mseTrain = MSE(theta, XtrainMulti, Ytrain)
print("Mse Training Error: ",mseTrain)
mseTest = MSE(theta, XtestMulti, Ytest)
print("Mse Test Error: ", mseTest)
```

5. Train Data Accuracy: 0.9226

Test Data Accuracy: 0.85632

Code:

```
def featureSVM(datum):
    feat = []
    feat.append(datum['beer/ABV'])
    feat.append(datum['review/taste'])
    return feat

XtrainSVM = [featureSVM(d) for d in trainData]
XtestSVM = [featureSVM(d) for d in testData]

YtrainSVM = [d['beer/style'] == 'American IPA' for d in trainData]
YtestSVM = [d['beer/style'] == 'American IPA' for d in testData]

clf = svm.SVC(C=1000, kernel = 'linear')
clf.fit(XtrainSVM, YtrainSVM)

train_predictions = clf.predict(XtrainSVM)
test_predictions = clf.predict(XtestSVM)

print("Train Data Accuracy: ",accuracy_score(YtrainSVM, train_predictions))
print("Test Data Accuracy: ", accuracy_score(YtestSVM, test_predictions))
```

6. New Feature Vector = [`beer/ABV'; `review/taste', If beer name contains IPA, If citrus exists in review text]

I looked at the common words in the reviews and almost every review had citrus/fruit word in it which I thought was a valuable way to differentiate between American IPA and others. Also, most beers had IPA in their name as well.

Train Data Accuracy: 0.96904

Test Data Accuracy: 0.9636

Code:

```
def featureSVM(datum):
    feat = []
    feat.append(datum['beer/ABV'])
    feat.append(datum['review/taste'])
    if('IPA' in datum['beer/name']):
        feat.append(1)
    else:
        feat.append(0)
    if('citrus' in datum['review/text']):
        feat.append(1)
    else:
        feat.append(0)
    return feat

XtrainSVM = [featureSVM(d) for d in trainData]
XtestSVM = [featureSVM(d) for d in testData]

YtrainSVM = [d['beer/style'] == 'American IPA' for d in trainData]
YtestSVM = [d['beer/style'] == 'American IPA' for d in testData]
```

```
clf = svm.SVC(C=1000)
clf.fit(XtrainSVM, YtrainSVM)

train_predictions = clf.predict(XtrainSVM)
test_predictions = clf.predict(XtestSVM)

print("Train Data Accuracy: ",accuracy_score(YtrainSVM, train_predictions))
print("Test Data Accuracy: ", accuracy_score(YtestSVM, test_predictions))
```

7. C maintains the balance between complexity of the model and is used to reach the sweet spot between underfitting and overfitting.

C	Train Data Accuracy	Test Data Accuracy
0.1	0.96784	0.96308
10	0.96884	0. 96268
1000	0. 96904	0.9636
100000	0. 97096	0. 95264

8. **Final log likelihood** = -6787.16668422

Train Data Accuracy: 0.91348

Test Data Accuracy: 0.9218

Code:

```
import numpy as np
import urllib.request
import scipy.optimize
from sklearn.metrics import accuracy_score
import random
from math import exp
from math import log

def getData(dataUrl):
    for line in urllib.request.urlopen(dataUrl):
        #print(line)
        yield eval(line)

print("Reading data")
data = list(getData("http://jmcauley.ucsd.edu/cse258/data/beer/beer_50000.json"))
print(data[0])

def inner(x,y):
  return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
  return 1.0 / (1 + exp(-x))

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
  loglikelihood = 0
  for i in range(len(X)):
    logit = inner(X[i], theta)
    loglikelihood -= log(1 + exp(-logit))
    if not y[i]:
      loglikelihood -= logit
  for k in range(len(theta)):
```

```
    loglikelihood -= lam * theta[k]*theta[k]
  print("ll =", loglikelihood)
  return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
  dl = [0.0]*len(theta)
  product = 1
  sum1 = 0;
  for k in range(len(dl)):
      for i in range(len(X)):
          logit = inner(X[i], theta)
          sum1 += X[i][k] * (1 - sigmoid(logit))
          if not y[i]:
              sum1 -= X[i][k]
      sum1-=(2*lam*theta[k])
      dl[k] = sum1
  # Negate the return value since we're doing gradient *ascent*
  return np.array([-x for x in dl])

trainData, testData = np.array_split(data, 2)

def featureSVM(datum):
    feat = []
    feat.append(datum['beer/ABV'])
    feat.append(datum['review/taste'])
    return feat

XtrainSVM = [featureSVM(d) for d in trainData]
XtestSVM = [featureSVM(d) for d in testData]

YtrainSVM = [d['beer/style'] == 'American IPA' for d in trainData]
YtestSVM = [d['beer/style'] == 'American IPA' for d in testData]

theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(XtrainSVM[0]), fprime, args =
(XtrainSVM, YtrainSVM, 1.0))
print("Final log likelihood =", -l)
print(theta)

YTrainPredictions = [inner(d, theta)>0 for d in XtrainSVM]
YTestPredictions = [inner(d, theta)>0 for d in XtestSVM]

print("Train Data Accuracy: ",accuracy_score(YtrainSVM, YTrainPredictions))
print("Test Data Accuracy: ", accuracy_score(YtestSVM, YTestPredictions))

print("End of program")
```