CSC2515
Machine Learning & Data Mining
Assignment 1

Tianxiang Chen
999473181

1. Learning basics of regression in Python

- Load the Boston housing data from the sklearn datasets module

The code is mainly provided in the skeleton code.
I showed some below. For more details, please check my q1.py file.

```
def load_data():
    boston = datasets.load_boston()
    X = boston.data
    y = boston.target
    features = boston.feature_names
    return X,y,features
```

Figure 1 Code for loading data

- Describe and summarize the data in terms of number of data points, dimensions, target, etc.

"X", the data set, is a (506, 13) matrix, which has 506 data. Each datum has 13 features, named as: 'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT'. Notice at this step, bias term hasn't been added.

"y", the target vector, is a (506,1) matrix. It has 506 predicted housing prices, which matches the 506 data for the input from "X".

For my own curiosity also for a better understanding of the features, I searched online for the full name of these features, as shown below [1]:

1	CRIM	Crime rate
2	ZN	Percentage of residential land zoned for lots over 25,000 ft^2
3	INDUS	Percentage of land occupied by nonretail business
4	CHAS	Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5	NOX	Nitric oxides concentration (parts per 10 million)
6	RM	Average number of rooms per dwelling
7	AGE	Percentage of owner-occupied units built prior to 1940
8	DIS	Weighted distances to five Boston employment centres
9	RAD	Index of accessibility to radial highways
10	TAX	Full-value property-tax rate per $10,000
11	PTRATIO	Pupil/teacher ratio by town
12	B	$1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
13	LSTAT	% Lower status of the population

Table I Features Explanation

- Visualization: present a single grid containing plots for each feature against the target. Choose the appropriate axis for dependent vs. independent variables. Hint: use *pyplot.tight* layout function to make your grid readable

I plotted each feature vs. price, altogether 13 plots. Notice in these plots, I have not normalized each feature. For later part where we compare the importance of each feature, I normalized them since we want to compare in a same way.

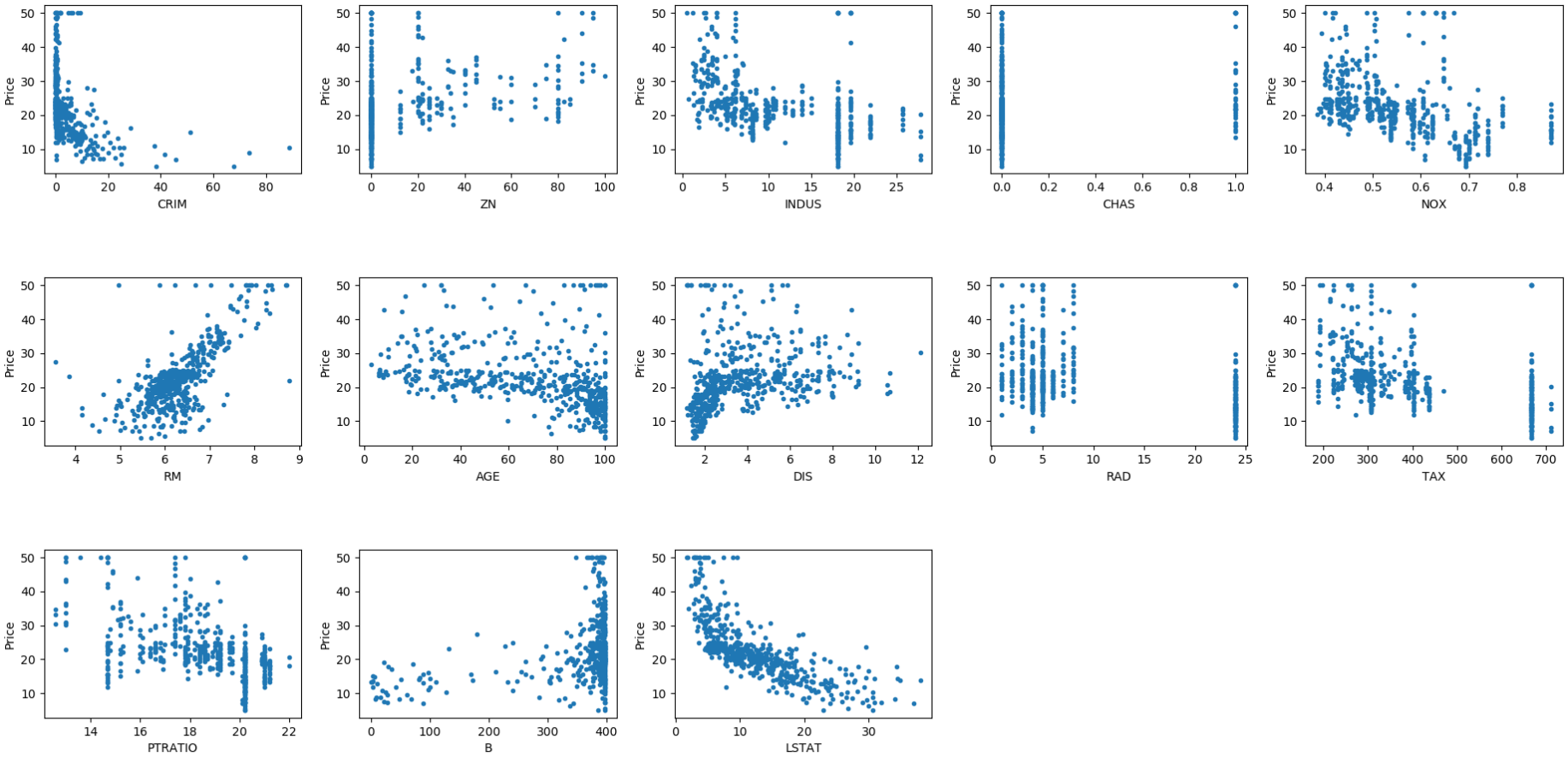

Figure 2 Plots for Price vs. each feature

- Divide your data into training and test sets, where the training set consists of 80% of the data points (chosen at random). Hint: You may find *numpy.random.choice* useful

As suggested, I used *numpy.random.choice* function.
I showed some code below. For more details, please check my q1.py file.

```
#TODO: Split data into train and test
i_train = np.random.choice(len(X), int(len(X)*0.8), replace = False)
i_train = sorted(i_train)
i = range(len(X))
i_test = list(set(i) - set(i_train))

# Separate into two sets
X_train = X[i_train,:]
y_train = y[i_train]
X_test = X[i_test,:]
y_test = y[i_test]
```

Figure 3 Code for split data into train and test

- Write code to perform linear regression to predict the targets using the training data. Remember to add a bias term to your model.

Bias term was added first by appending a column of zeros to the first column of the X matrix. The w matrix is solved by using *np.linalo.solve* function, using the equation learned in the lecture:

$$w^* = (X^T X)^{-1} Xy$$

```
def fit_regression(X,Y):
    #TODO: implement linear regression
    # Add bias term
    col_ones = np.ones(len(X))
    X=np.column_stack((col_ones,X))
    # Remember to use np.linalg.solve instead of inverting!
    w_opt = np.linalg.solve(np.dot(X.transpose(), X), np.dot(X.transpose(), Y))
    return w_opt
```

Figure 4 Code for implementing linear regression

- Tabulate each feature along with its associated weight and present them in a table. Explain what the sign of the weight means in the third column ('INDUS') of this table. Does the sign match what you expected? Why?

Notice since we randomly select data set into training and test sets, each time weight for each feature varies. Here I ran the code 3 times and recorded down the feature values. The third column, ('INDUS'), corresponds to w3 here, which is highlight in red**.** Also, since we want to compare the w magnitude to rank/value the features, we should normalize features to let them have the same weight (I make every feature in the range of 0-1 by dividing its max. value among 506 input data, shown in below)

```
#Normalize the data(X), for later comparing w
for i in range(X.shape[1]):
    col_max = X[:,i].max()
    X[:,i] /= col_max
```

Figure 5 Code for normalizing features within [0,1] range

I noticed the sign of INDUS varies. Also, the w3's magnitude is quite small compared with other w coefficients.

This result matches what my expected.

This could be verified with the INDUS vs. Price plot shown in previous section. In that plot, INDUS doesn't show a direct relation to the price. Price can be either high or low when the INDUS is small. The same applies to price when INDUS is large.

So, from the plot, we could say INDUS has an insignificant impact on price (w3 varies in sign, with a small absolute magnitude).

	1st time	2nd time	3rd time
w0 (Bias)	41.95	35.42	36.86
w1	-9.64	-8.74	-7.25
w2	4.89	3.94	3.65
w3	0.62	-0.47	1.27
w4	2.62	1.56	2.58
w5	-15.52	-13.46	-15.56
w6	28.45	32.25	34.82
w7	0.79	0.07	-0.77
w8	-18.60	-16.78	-16.45
w9	9.37	7.09	7.39
w10	-11.21	-8.43	-9.46
w11	-21.84	-20.41	-21.77
w12	3.75	3.68	3.35
w13	-22.81	-19.38	-20.28

Table II Feature Coefficients

- Test the fitted model on your test set and calculate the Mean Square Error of the result.
- Suggest and calculate two more error measurement metrics; justify your choice.

I prefer to answer these two questions together.

I chose Root Mean Square Error and Mean Absolute Error. Compared with Mean Square Error, these two error measurement metrics shows the error averaged for each individual.

I used the result run in the previous part (I also recorded the errors for that 3 runs).

	Mean Square Error	Root Mean Square Error	Mean Absolute Error
Run 1	20.079	4.481	3.272
Run 2	28.460	5.335	3.676
Run 3	21.629	4.651	2.995
Average	23.39	4.82	3.31

Table III Error Measurement Metrics

I also plot $\hat{y}$ (test) vs. y(predicted) for one run. The red line here means $y_{test}=y_{pred}$. So, the closer the data points(blue) locates to the red line, the better the linear regression preforms.

Figure 6 Prices vs. Predicted Prices

- Feature Selection: Based on your results, what are the most significant features that best predict the price? Justify your answer.

In my w (Table 1 in previous section), except w0(bias), these features are the most significant features, ranked by their magnitude with sign indicated:

w6(+) > w11(-) ≈ w13(-) >w8(-) >others

The result here is reasonable to me.

w6 is RM, "Average number of rooms per dwelling", with a positive sign. It is obvious the more room a house has, the higher the house's value is.

w11(PTRATIO) and w13(LSTAT), "Pupil/teacher ratio by town" and "% Lower status of the population", are the second and third crucial factor for the house price, both with a negative sign. This is also reasonable that the house in the neighbourhood which has good education resources or high status (rich people living region I guess?) will have a higher price.

w8(dis), "Weighted distances to five Boston employment centres", is the fourth key factor for the house price, with a negative sign. It indicates that the farer the house locates to the employment centers, the cheaper the house values.

2. Locally reweighted regression

2.1. We are given: $w^* = \arg\min \frac{1}{2}\sum_{i=1}^{N} a^{(i)}(y^{(i)} - w^T x^{(i)})^2 + \frac{1}{2}\| w \|^2$

Firstly, we can rewrite the error function in matrix form:

$$J(w) = \frac{1}{2} A(y - Xw)(y - Xw)^T + \frac{\lambda}{2} w^T wI$$

Then, simplification was made for the equation:

$$J(w) = \frac{1}{2} A(yy^T - (Xw)y^T - y(Xw)^T + Xw(Xw)^T) + \frac{\lambda}{2} w^T wI$$

$$J(w) = \frac{1}{2} A(yy^T - 2 * Xwy^T + Xww^T X^T) + \frac{\lambda}{2} w^T wI$$

Taking the derivative of w and set to zero, we found:

$$\frac{\partial J(w)}{\partial w} = \frac{1}{2}(2X^T AXw - 2 * X^T Ay) + \lambda w = 0$$

$$(X^T AX + \lambda I)w = X^T Ay$$

So,

$$w* = (X^T AX + \lambda I)^{-1} X^T Ay$$

Proved.

2.2 In order to improve the efficiency (I improved my runtime from 14min38sec to 66sec), I ran the l2 function in run_on_fold instead of LRLS, which calculates all the distances into one big matrix M. Later for each LRLS, one row of the M was added to pass into LRLS. This avoids to calling l2 function repeatedly in LRLS function, which improves the runtime.
I didn't use the logsumexp function. (I put logsumexp approach in comment in my code) From my test, the code I wrote leads to the same result as using logsumexp while mine runs much faster.

```
for j,tau in enumerate(taus):
    M = np.exp(-1 * l2(x_test, x_train) / (2*(tau**2)))
    # M = -1 * l2(x_test, x_train) / (2*(tau**2))
    predictions =  np.array([LRLS(x_test[i,:],x_train,y_train, tau, M[i,:]) \
                   for i in range(N_test)])
    losses[j] = ((predictions.flatten()-y_test.flatten())**2).mean()
    print(j+1, "/", len(taus))
```

Figure 7 Calculate all distances into M in run_on_fold instead of LRLS

2.3 The plot shows the average loss for different values of τ in the range [10,1000]

The min loss I got is 11.9355.

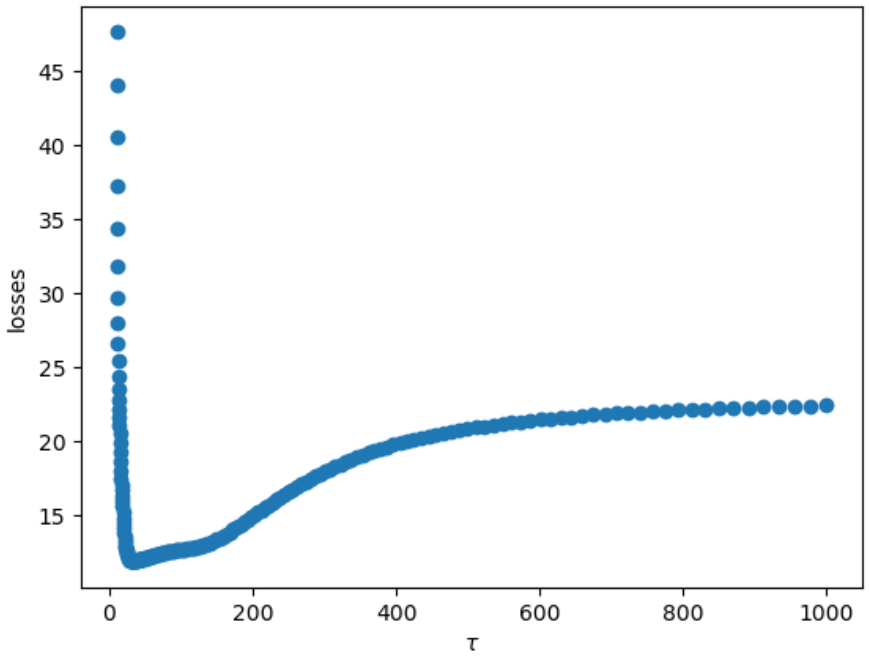

Figure 8 Average Loss for different values of τ in the range [10,1000]

2.4 How does this algorithm behave when $\tau \to \infty$? When $\tau \to 0$?

$\tau \to \infty$ means the Gaussian is with a large variance so the influence of x_i is more. In this case, even the distance between x and x_i is large, the x_i will still have influence on deciding the class of the new testing datum x.
When $\tau \to 0$, the small variance implies the x_i won't have a widespread influence on the new testing datum x.
So, $\tau \to 0$ will lead to a high bias and low variance model. In this case, the loss will be very large. And $\tau \to \infty$ leads to a low bias but high variance model. In this case, matrix A act as it does not exist and the loss will be relative small. There will be a point where $\tau \in (0, \infty)$, which takes the influence of x_i but not overtakes it and makes the loss minimum.

3. Mini-batch SGD Gradient Estimator

3.1 Assume we call one possible draw of the mini-batch $M = \frac{1}{m}\sum_{i \in I} a_i$. When we want to pick m ($m \leqslant n$) from a n data set as one draw, we have C_n^m choices. This means we have C_n^m different M, with each one has an equal probability to occur ($p_i = \frac{1}{C_n^m}$).

So, we can rewrite the left-hand side of the equation as:

$$E_I[\frac{1}{m}\sum_{i \in I} a_i] = \sum_{i=1}^{C_n^m} M_i p_i \text{ with } p_i = \frac{1}{C_n^m}$$

$$= \sum_{i=1}^{C_n^m} \frac{M_i}{C_n^m}$$

$$= \frac{\sum_{i=1}^{C_n^m} M_i}{C_n^m}$$

For the denominator, we know that $C_n^m = \frac{n!}{m!(n-m)!}$. Now, we are left with the numerator. We know M can only have data within the whole data set. So, thinking it in another way, each datum in the set $\{a_1, a_2, \ldots a_n\}$ will occur C_{n-1}^{m-1} times. (assuming we've already picked a_i, left n-1 data to fill in m-1 positions)

Now the left-hand side can be further rewritten as:

$$\frac{\sum_{i=1}^{C_n^m} M_i}{C_n^m} = \frac{\frac{1}{m} C_{n-1}^{m-1} \sum_{i=1}^{n} a_i}{C_n^m} = (\frac{1}{m}\sum_{i=1}^{n} a_i) \frac{\frac{(n-1)!}{(m-1)!(n-m)!}}{\frac{n!}{m!(n-m)!}} = (\frac{1}{m}\sum_{i=1}^{n} a_i) \frac{m!(n-1)!}{n!(m-1)!}$$

$$= (\frac{1}{m}\sum_{i=1}^{n} a_i) \frac{m}{n} = \frac{1}{n}\sum_{i=1}^{n} a_i = RHS$$

Proved.

3.2 From the question, we know loss function is defined as:

$$L(x, y, \theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(x^{(i)}, y^{(i)}, \theta)$$

For the gradient, we have:

$$\nabla \ell(x^{(i)}, y^{(i)}, \theta) = \frac{\partial \ell(x^{(i)}, y^{(i)}, \theta)}{\partial w}$$

So, for each i, the gradient of the loss function will be a vector. The whole gradient set $\nabla \ell(x, y, \theta)$ will be a vector set. So, it has a similar form as the one in q3.1. Here, we can consider a set of vectors $\{a_1, a_2, \ldots a_n\}$, where $a_i = \nabla \ell(x^{(i)}, y^{(i)}, \theta)$.

Thus, with the conclusion from q3.1, we can say:

$$E_I[\nabla L(x, y, \theta)] = E_I[\frac{1}{m}\sum_{i \in I} \ell(x^{(i)}, y^{(i)}, \theta)] = \frac{1}{n}\sum_{i=1}^{n} \ell(x^{(i)}, y^{(i)}, \theta) = \nabla L(x, y, \theta)$$

$$\therefore E_I[\nabla L(x, y, \theta)] = \nabla L(x, y, \theta)$$

Proved.

3.3 The result shows that with Mini-batch SGD Gradient Estimator, we can still get a quite accurate result, with a shorter computation time compared with the true gradient approach.

3.4 Firstly, I would like to write it in scalar form and then convert it into matrix form.
For an individual $y^{(i)}$, its mapping function can be expressed as:

$$y^{(i)} = w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots w_{13} x_1^{(i)}$$

Where w_0 is the bias term.

So, for an individual $y^{(i)}$, its cost function is:

$$(y^{(i)} - x^{(i)})^2 = (y^{(i)} - (w_0 x_0^{(i)} + w_1 x_1^{(i)} + \ldots w_{13} x_1^{(i)}))^2$$

When taking derivative with w_0, $w_1 \ldots w_i$, we can obtain:

$$\frac{\partial (y^{(i)} - x^{(i)})^2}{\partial w_0} = -2(y^{(i)} - (w_0 x_0^{(i)} + w_1 x_1^{(i)} + ... w_{13} x_1^{(i)})) * x_0^{(i)}$$

$$\frac{\partial (y^{(i)} - x^{(i)})^2}{\partial w_1} = -2(y^{(i)} - (w_0 x_0^{(i)} + w_1 x_1^{(i)} + ... w_{13} x_1^{(i)})) * x_1^{(i)}$$

$$\vdots$$

$$\frac{\partial (y^{(i)} - x^{(i)})^2}{\partial w_i} = -2(y^{(i)} - (w_0 x_0^{(i)} + w_1 x_1^{(i)} + ... w_{13} x_1^{(i)})) * x_i^{(i)}$$

Thus, rewriting it into matrix form:

$$\frac{\partial \ell(x, y, w)}{\partial w} = -2x^T (y - xw)$$

Notice that the question asks for x not "X", so it is just an input data set, the same as what I wrote here.

3.5 Code was written to calculate the gradient using true gradient approach and min-batch approach. Notice in the code, I changed the skeleton code by adding a variable m which is defined as the mini-batch size.

The linear regression gradient function is written as below:

```
def lin_reg_gradient(X, y, w, m): #Add m here for batch size
    '''
    Compute gradient of linear regression model parameterized by w
    '''
    batch_grad = -2 * np.dot(X.transpose() , (y - np.dot(X,w))) / m

    return batch_grad
```

Figure 9 Linear Regression Gradient Function

Then, for true gradient, we just passed m as the true data set (506 in this case). And for mini-batch, m was set as 50 and we ran the function 500 times (as required) then averaged its value. For mini-batch approach, we need to set w to the same initial status every time as the true gradient value. This makes the two approaches have the same initial point so make it reasonable for the later comparison.

```
# Traditional way to calculate gradient
batch_grad_old = np.zeros(w.shape[0])
batch_grad_old = lin_reg_gradient(X, y, w, len(X))
print 'True gradient: \n', batch_grad_old
```

Figure 10 True Gradient

```
# Use mini-batch to calculate gradient
batch_grad_new = np.zeros(w.shape[0])
for i in range(Round):
    # Initialize w again, using the same w as true gradient
    w = w_backup
    X_b, y_b = batch_sampler.get_batch()
    batch_grad_new += lin_reg_gradient(X_b, y_b, w, BATCHES)/Round
print 'Gradient using mini-batch:\n', batch_grad_new
```

Figure 11 Mini-Batch Gradient

As required in the handout, I used both the squared distance metric and cosine similarity to compare the true gradient and mini-batch gradient.

I still ran the code three times and the result was recorded as:

	Squared Distance	Cosine Similarity
Run 1	99285.6	1.000000
Run 2	4225297.1	0.999996
Run 3	240196.6	1.000000

Table IV 3 Runs Result for Squared Distance and Cosine Similarity

So, I found the squared distance is quite big. This is due to in this data set, the values in the gradient vector was very large. Cosine Similarity in this case would be a better way to compare the two variable/result is similar in direction or not. From all three, though the square distance is large, the cosine similarity is all close to one, which shows the true gradient and mini-batch gradient point to similar directions. (1=cos (0°), so the two variable is almost the same direction)

3.6 I randomly chose a w (w[5]). The variance σ vs. m$\in$ [0,400] was plotted below, in log scale.

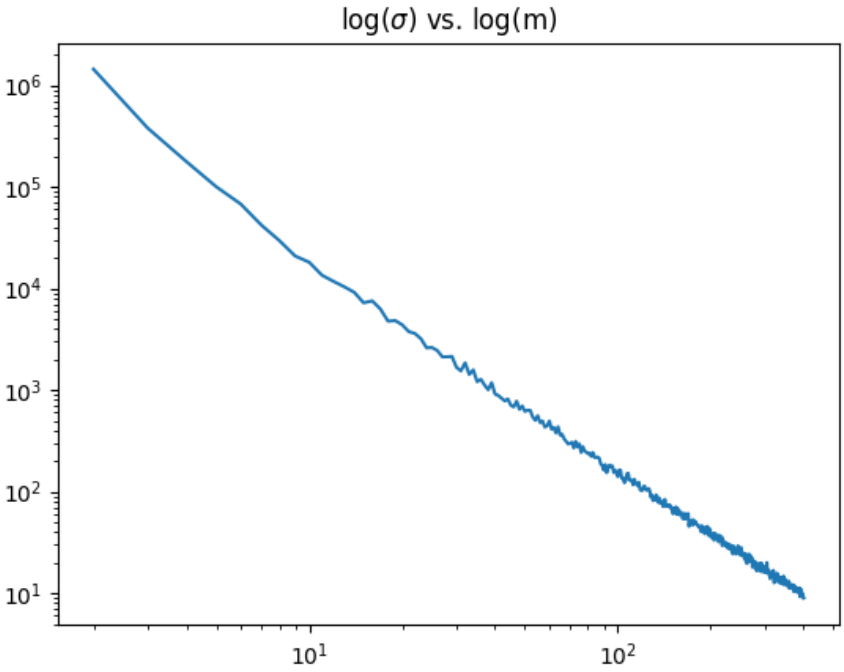

Figure 12 log(σ) vs. log(m)

References

[1] G. Shmueli, N. Patel and P. Bruce, *Data mining for business intelligence*. Hoboken: Wiley, 2007, p. 36.