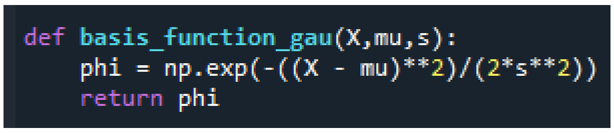
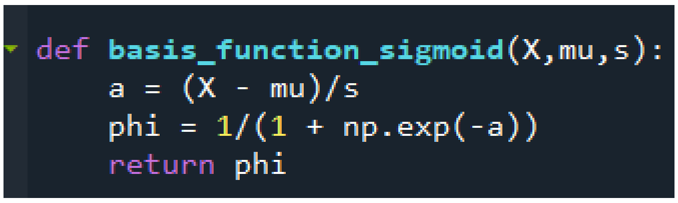
HW3 Report

1. **Please use Maximum Likelihood and Least Squares to train the model. Then, use your trained linear model to predict the burnt calories and compute the mean squared error for each data in testing set.**

First, set up my basis function, which is Gaussian function and sigmoid function.



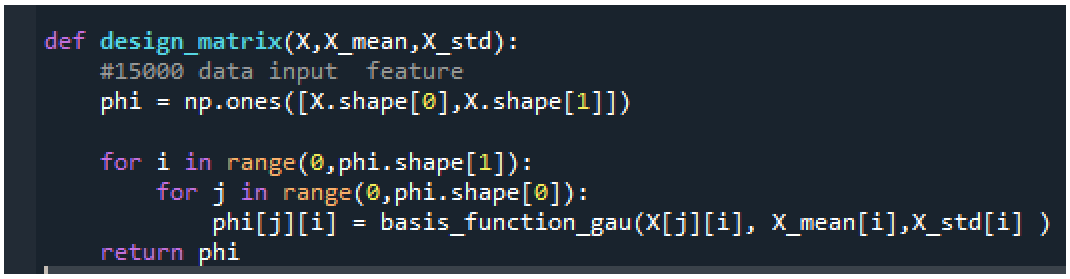
```python
def basis_function_gau(X,mu,s):
    phi = np.exp(-((X - mu)**2)/(2*s**2))
    return phi
```



```python
def basis_function_sigmoid(X,mu,s):
    a = (X - mu)/s
    phi = 1/(1 + np.exp(-a))
    return phi
```

Set up the design matrix from the book.

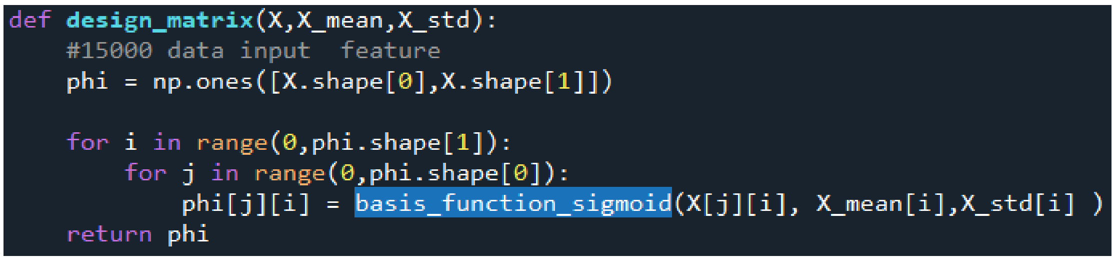
Gaussian



```python
def design_matrix(X,X_mean,X_std):
    #15000 data input  feature
    phi = np.ones([X.shape[0],X.shape[1]])

    for i in range(0,phi.shape[1]):
        for j in range(0,phi.shape[0]):
            phi[j][i] = basis_function_gau(X[j][i], X_mean[i],X_std[i] )
    return phi
```

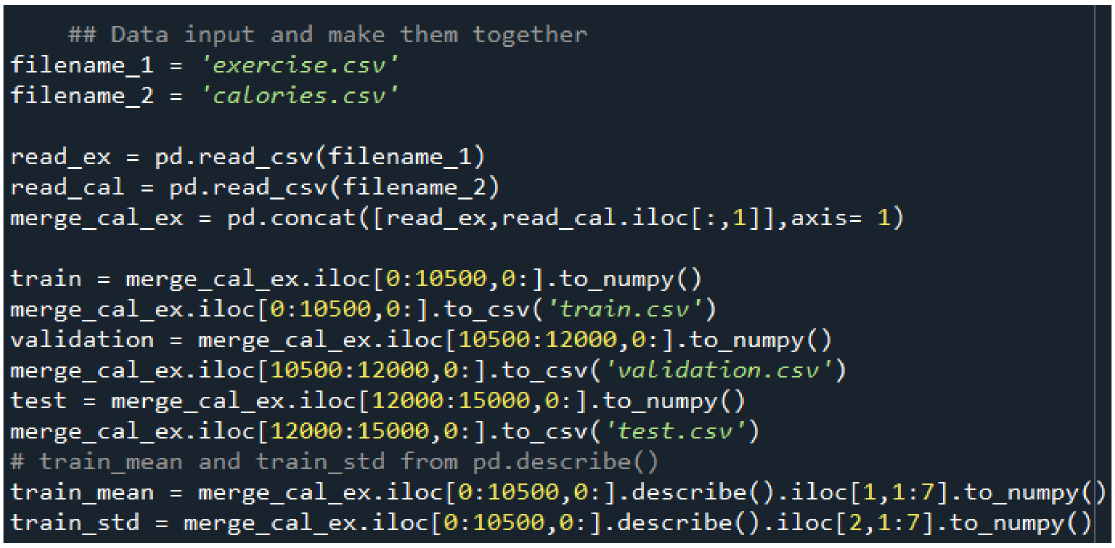
Sigmoid



```python
def design_matrix(X,X_mean,X_std):
    #15000 data input  feature
    phi = np.ones([X.shape[0],X.shape[1]])

    for i in range(0,phi.shape[1]):
        for j in range(0,phi.shape[0]):
            phi[j][i] = basis_function_sigmoid(X[j][i], X_mean[i],X_std[i] )
    return phi
```

Next step, merge them and split them into three part. Training, validation and testing.

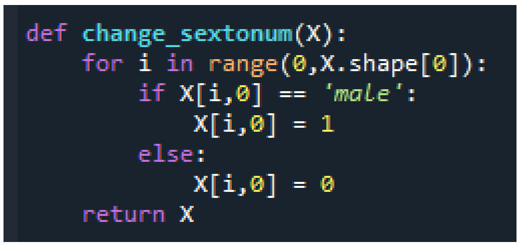


```
    ## Data input and make them together
filename_1 = 'exercise.csv'
filename_2 = 'calories.csv'

read_ex = pd.read_csv(filename_1)
read_cal = pd.read_csv(filename_2)
merge_cal_ex = pd.concat([read_ex,read_cal.iloc[:,1]],axis= 1)

train = merge_cal_ex.iloc[0:10500,0:].to_numpy()
merge_cal_ex.iloc[0:10500,0:].to_csv('train.csv')
validation = merge_cal_ex.iloc[10500:12000,0:].to_numpy()
merge_cal_ex.iloc[10500:12000,0:].to_csv('validation.csv')
test = merge_cal_ex.iloc[12000:15000,0:].to_numpy()
merge_cal_ex.iloc[12000:15000,0:].to_csv('test.csv')
# train_mean and train_std from pd.describe()
train_mean = merge_cal_ex.iloc[0:10500,0:].describe().iloc[1,1:7].to_numpy()
train_std = merge_cal_ex.iloc[0:10500,0:].describe().iloc[2,1:7].to_numpy()
```

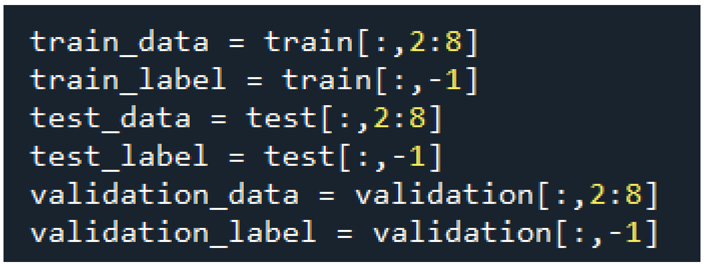
In addition, I use the function to change male to 1 and female to 0.



```
def change_sextonum(X):
    for i in range(0,X.shape[0]):
        if X[i,0] == 'male':
            X[i,0] = 1
        else:
            X[i,0] = 0
    return X
```

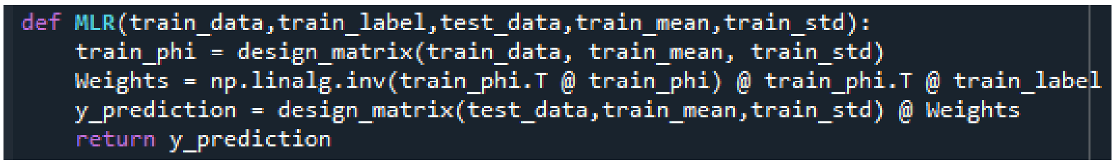
Splitting the data.

Feature-only use: Age, Height, Weight, Duration, Heart rate, Body temperature.

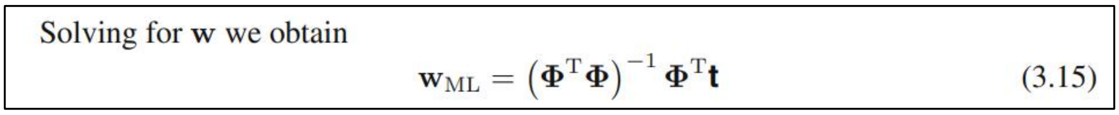


```
train_data = train[:,2:8]
train_label = train[:,-1]
test_data = test[:,2:8]
test_label = test[:,-1]
validation_data = validation[:,2:8]
validation_label = validation[:,-1]
```

Define the Maximum linear regression,



```
def MLR(train_data,train_label,test_data,train_mean,train_std):
    train_phi = design_matrix(train_data, train_mean, train_std)
    Weights = np.linalg.inv(train_phi.T @ train_phi) @ train_phi.T @ train_label
    y_prediction = design_matrix(test_data,train_mean,train_std) @ Weights
    return y_prediction
```



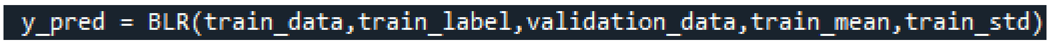
Solving for $\mathbf{w}$ we obtain

$$\mathbf{w}_{\mathrm{ML}} = \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \tag{3.15}$$

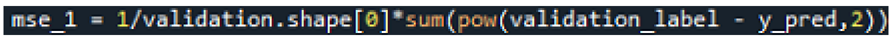
We can calculate the weights by the above figure. The phi function is defined by the last step.

Finally, it's a prediction for my test data.

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = \mathbf{W}^{\mathrm{T}} \phi(\mathbf{x}) \tag{3.31}$$

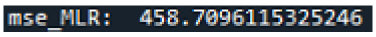


```
 y_pred = BLR(train_data,train_label,validation_data,train_mean,train_std)
```



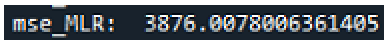
```
mse_1 = 1/validation.shape[0]*sum(pow(validation_label - y_pred,2))
```

So the MSE with the sigmoid answer is



```
mse_MLR:  458.7096115325246
```

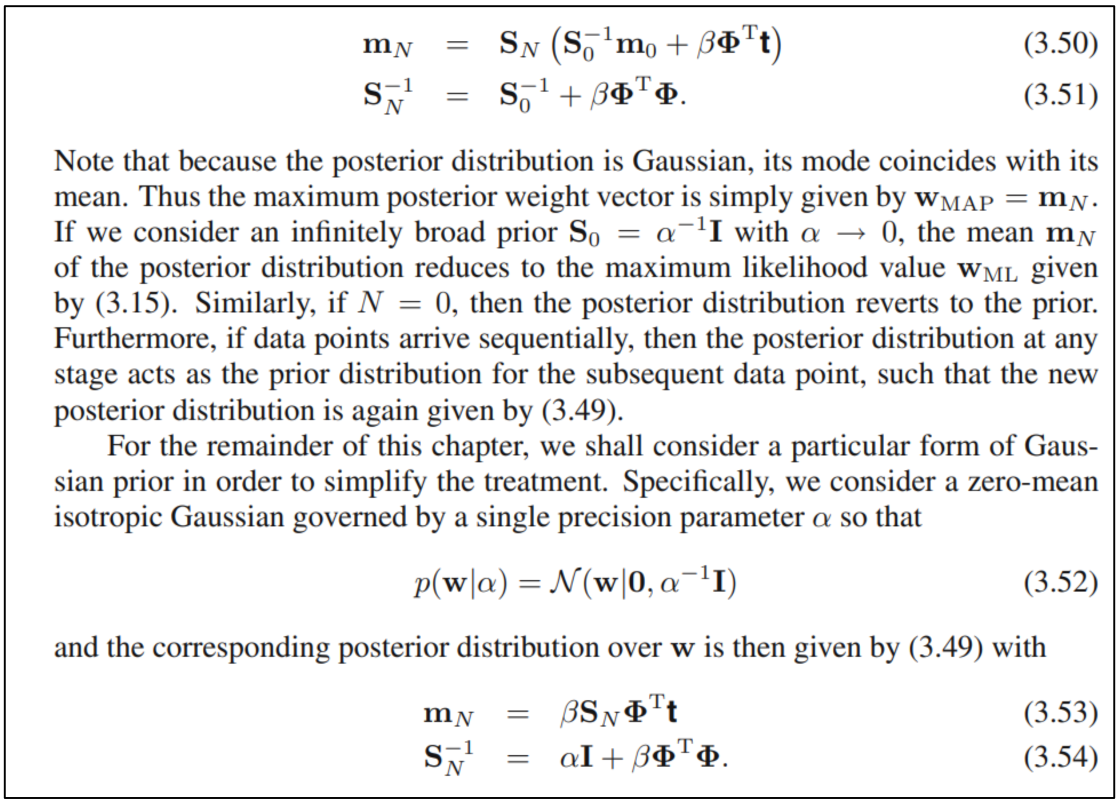
the MSE with the gaussian answer is



```
mse_MLR:  3876.0078006361405
```

We can take an easy summary, sigmoid is better for this problem.

2. **Please use Bayesian Linear Regression to estimate w. Then, use your estimated parameter to predict the burnt calories and compute the mean squared error for each data in testing_set.**



$$\mathbf{m}_N = \mathbf{S}_N \left(\mathbf{S}_0^{-1}\mathbf{m}_0 + \beta\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}\right) \tag{3.50}$$
$$\mathbf{S}_N^{-1} = \mathbf{S}_0^{-1} + \beta\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}. \tag{3.51}$$

Note that because the posterior distribution is Gaussian, its mode coincides with its mean. Thus the maximum posterior weight vector is simply given by $\mathbf{w}_{\mathrm{MAP}} = \mathbf{m}_N$. If we consider an infinitely broad prior $\mathbf{S}_0 = \alpha^{-1}\mathbf{I}$ with $\alpha \to 0$, the mean $\mathbf{m}_N$ of the posterior distribution reduces to the maximum likelihood value $\mathbf{w}_{\mathrm{ML}}$ given by (3.15). Similarly, if $N = 0$, then the posterior distribution reverts to the prior. Furthermore, if data points arrive sequentially, then the posterior distribution at any stage acts as the prior distribution for the subsequent data point, such that the new posterior distribution is again given by (3.49).

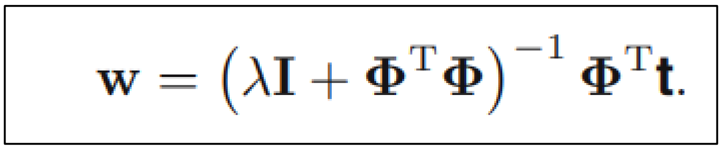
For the remainder of this chapter, we shall consider a particular form of Gaussian prior in order to simplify the treatment. Specifically, we consider a zero-mean isotropic Gaussian governed by a single precision parameter α so that

$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I}) \tag{3.52}$$

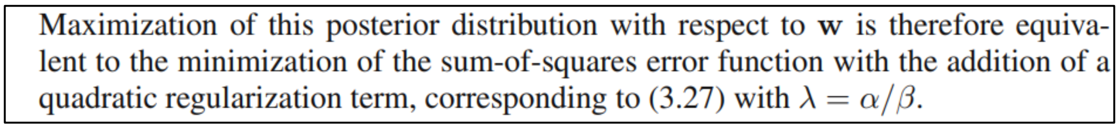
and the corresponding posterior distribution over $\mathbf{w}$ is then given by (3.49) with

$$\mathbf{m}_N = \beta\mathbf{S}_N\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t} \tag{3.53}$$
$$\mathbf{S}_N^{-1} = \alpha\mathbf{I} + \beta\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}. \tag{3.54}$$

The figure shows that the weights are equal to the mean $\mathrm{m_N}$.
With several steps, we can derive the function in the following figure.

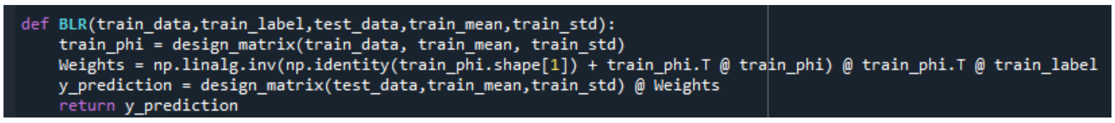


$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$



Maximization of this posterior distribution with respect to $\mathbf{w}$ is therefore equivalent to the minimization of the sum-of-squares error function with the addition of a quadratic regularization term, corresponding to (3.27) with $\lambda = \alpha/\beta$.

The alpha and beta are all the precision (inverse variance). When the precision becomes large, it means that the variance must be small. So I set up the alpha and beta to have the same magnitude.



```
def BLR(train_data,train_label,test_data,train_mean,train_std):
    train_phi = design_matrix(train_data, train_mean, train_std)
    Weights = np.linalg.inv(np.identity(train_phi.shape[1]) + train_phi.T @ train_phi) @ train_phi.T @ train_label
    y_prediction = design_matrix(test_data,train_mean,train_std) @ Weights
    return y_prediction
```

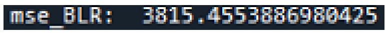
```
y_pred_MLR = MLR(train_data,train_label,test_data,train_mean,train_std)
```

```
mse_2 = 1/test.shape[0]*sum(pow(test_label - y_pred_MLR,2))
```

The sigmoid basis function with BLR is

```
mse_BLR:  441.0126052841631
```

The gaussian basis function with BLR is



```
mse_BLR:  3815.4553886980425
```

We can take an easy summary, sigmoid is better for this problem.

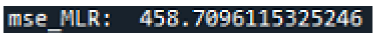
3. Please discuss the difference between Maximum Likelihood and Bayesian Linear Regression. Plot the best-fit lines for both models.

There is almost the same operation for the two method, and the only different part is $\lambda = \frac{\alpha}{\beta}$

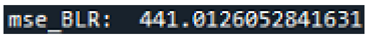
With the constant, the weights cannot become gigantic numbers due to inverse operations.

Moreover, it can avoid model overfitting because some of the weights constrain by the λ.

Compare to two methods MSE, Bayesian linear regression is smaller than maximum linear regression.



```
mse_MLR:  458.7096115325246
```



```
mse_BLR:  441.0126052841631
```

Finally, I plot the Duration figure with all validation data. (1500)

The blue dot line is ordinary least squares and it is the same as maximum likelihood regression.

Finally, the red line is Bayesian linear regression.

We can see that there are the same lines in the figure.

Why there are many red lines in the figure, I think that our intercept, slope, and sigma are all normal distributions.

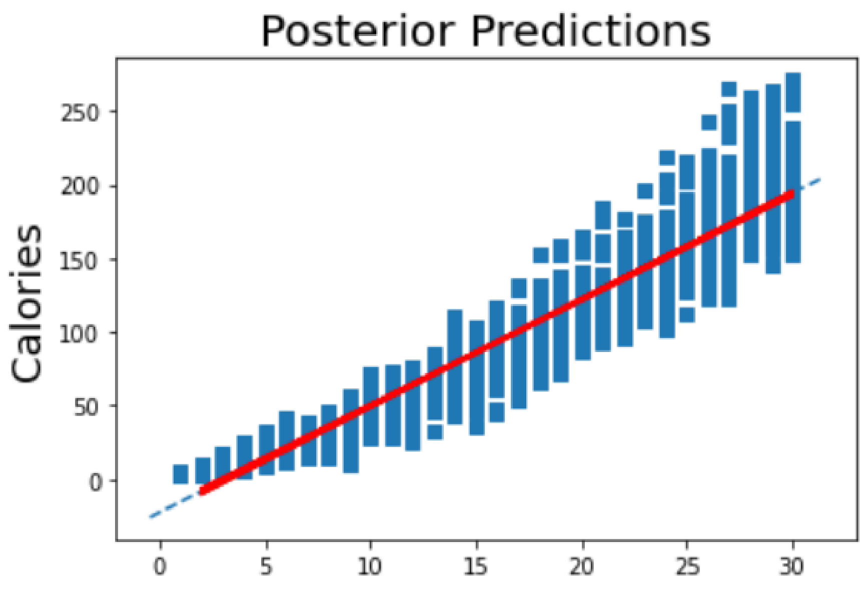
I sample it 100 times, so it will return 100 slopes and intercepts.

Finally, using the function shows all possible regression in the figure.

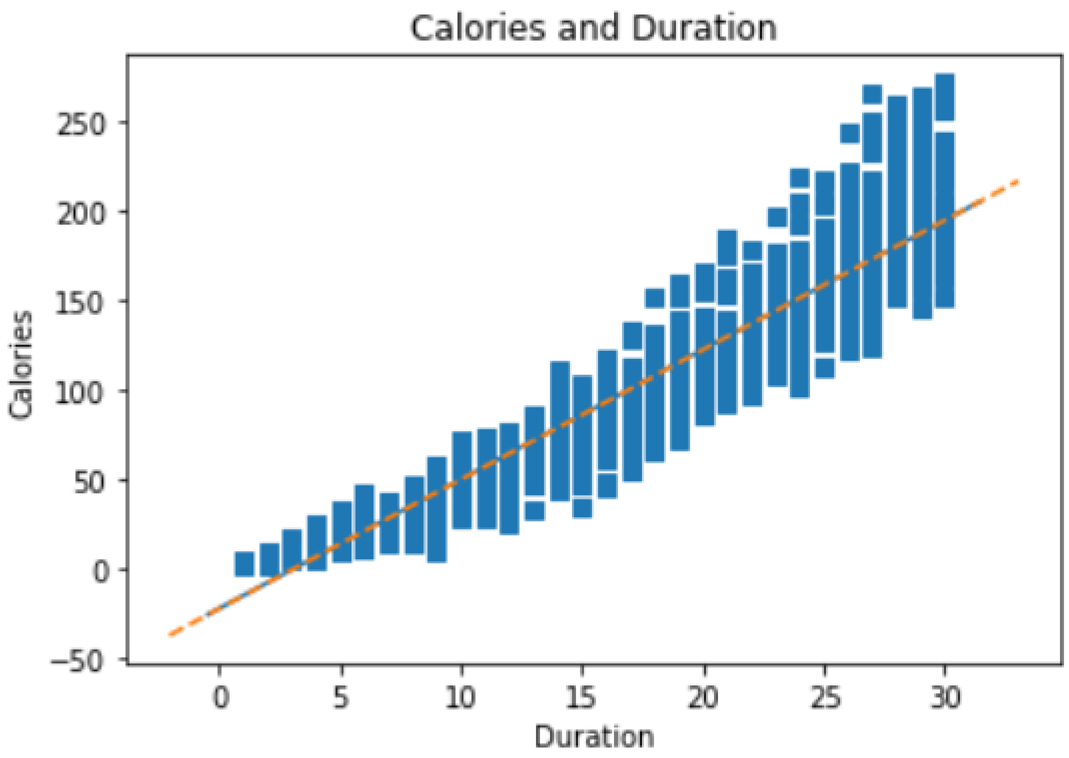
Also, I can get the figure by my own model.

Just take the weights from the two methods.

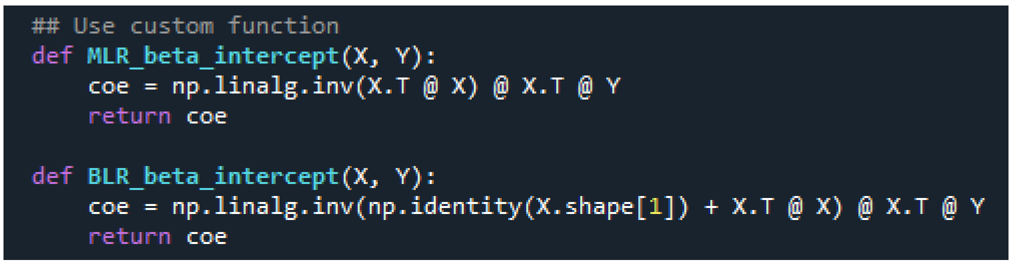
Finally, plot the data.



Source code: https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7#:~:text=In%20the%20Bayesian%20viewpoint%2C%20we,drawn%20from%20a%20probability%20distribution



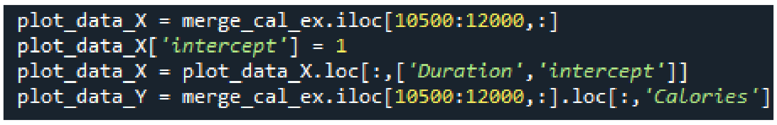
The upper image shows that the two method lines are almost the same.
Only need to take the weight in the code and add an intercept to make it find the feature weight and intercept magnitude.
Finally, I can build up the figure without any package usage.
The two figure lines' positions are the same, and it also presents that the book provides equations that are the same as the public packages.
The following code is my two custom functions.



```
## Use custom function
def MLR_beta_intercept(X, Y):
    coe = np.linalg.inv(X.T @ X) @ X.T @ Y
    return coe

def BLR_beta_intercept(X, Y):
    coe = np.linalg.inv(np.identity(X.shape[1]) + X.T @ X) @ X.T @ Y
    return coe
```

Need to set up an intercept column for the above function to generate intercept and slope.



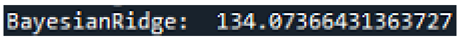
```
plot_data_X = merge_cal_ex.iloc[10500:12000,:]
plot_data_X['intercept'] = 1
plot_data_X = plot_data_X.loc[:,['Duration','intercept']]
plot_data_Y = merge_cal_ex.iloc[10500:12000,:].loc[:,'Calories']
```

4. Please implement any regression model you want to get the best possible MSE.

Source: https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

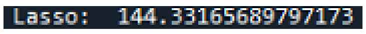
I use the different functions in the sklearn package.
The best answer I found is using ordinary least squares.

Bayesian ridge



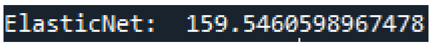
```
BayesianRidge:  134.07366431363727
```

Lasso function



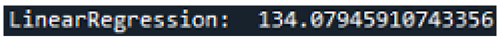
```
Lasso:  144.33165689797173
```

Elastic Net function



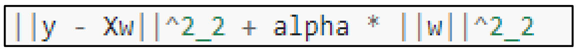
```
ElasticNet:  159.5460598967478
```

Linear Regression function (OLS)



```
LinearRegression:  134.07945910743356
```

It is the same from the question one and question two, because the Bayesian ridge has to take weights into consideration.
It will avoid overfitting and improve the models' generalization.



```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

It will use the L2 norm to avoid the weight become larger.
Source: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge