Part 2. Programming assignment : (70%)

1) (20 points for implementation + 10 points for MSE Screenshot)
Please use Least Squares i.e. Maximum Likelihood Estimation (see Q.3) 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.

In the beginning, I load exercise.csv and calories.csv, and convert gender from string description to binary, then normalize other features and the label calories, merge them together, and then split them into 70:10:20 for training, validation, and testing data.

To apply MLR, the process is as follows

Step 1: calculate gaussian basis, use formula from textbook 3.4

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

```
def gaussian_basis(X, mu, s):
    return np.exp(-((X-mu)**2 / (2*s**2)))                       # calculate gaussian basis (Textbook 3.4)
```

Step 2: calculate design matrix as Phi, use formula from textbook 3.16

$$\mathbf{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

```
def  Phi_matrix(X, X_mu, X_s):
    #print(X)
    #print(X_mu)
    #print(X_s)
    #print(X.shape)
    Phi = np.ones([X.shape[0], X.shape[1]])
    Phi[:, 0] = X[:, 0]
    for j in range(0, X.shape[1]):
        for i in range(0, X.shape[0]):
            Phi[i][j] = gaussian_basis(X[i][j], X_mu[j], X_s[j])          # calculate design matrix, Phi (Textbook 3.16)
    return Phi
```

Step 3: calculate w, use formula from textbook 3.15

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

Step 4: predict data, calculate Phi of test_data, then multiply Phi of test_data and w of train_data together from textbook 3.31, results are predicted values.

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

```
def MLR(train_data, train_label, test_data, train_mean, train_std):              # functions MLR()
    train_Phi = Phi_matrix(train_data, train_mean, train_std)
    #print(train_Phi)
    weights = np.linalg.inv(train_Phi.T @ train_Phi) @ train_Phi.T @ train_label     # calculate w (Textbook 3.15)
    y_pred = Phi_matrix(test_data, train_mean, train_std) @ weights                  # y_pred = Phi @ w (Textbook 3.31)
    return y_pred
```

Step 5: calculate MSE

```
mse_MLR = 1/len(test_data) * np.sum((test_label - y_pred_MLR)**2)                # calculate mse of MLR
```

RESULT:

```
mse_MLR:  0.034191017631912485
```

2) (20 points for implementation + 10 points for MSE Screenshot) 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 Validation_set.

The only difference between BLR and MLR is step 3, BLR has one more λI item from textbook 3.28

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

```
def BLR(train_data, train_label, test_data, train_mean, train_std):              # functions BLR()
    train_Phi = Phi_matrix(train_data, train_mean, train_std)
    #print(train_Phi)
    weights = np.linalg.inv(np.identity(train_Phi.shape[1]) + train_Phi.T @ train_Phi) @ train_Phi.T @ train_label          # calculate w (Textbook 3.28), lambda = 1
    y_pred = Phi_matrix(test_data, train_mean, train_std) @ weights                  # y_pred = Phi @ w (Textbook 3.31)
    return y_pred
```

RESULT:

```
mse_BLR:  0.03356227959809484
```

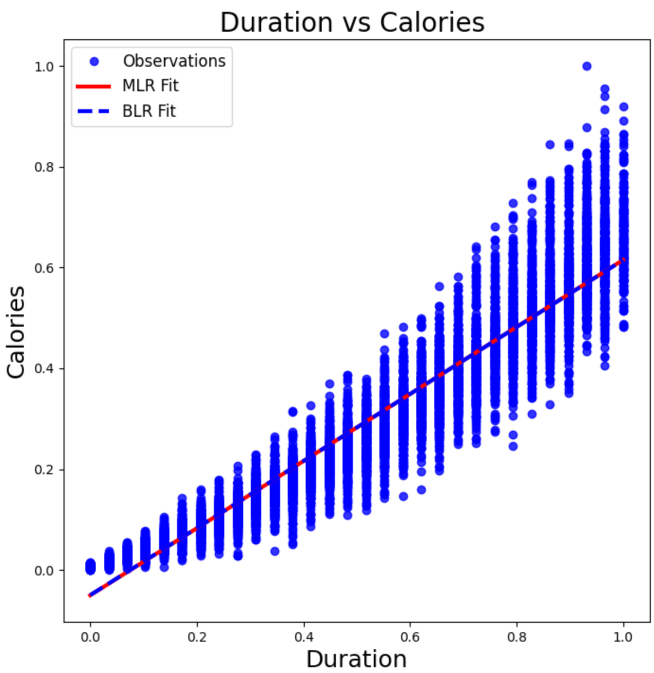
3) (10 points) Please discuss the difference between Maximum Likelihood and Bayesian Linear Regression. Plot the best fit lines for both models. Best fit lines for Bayesian Linear Regression means that you have to plot the intercept and slope.

In this part, I used my own calculation of the Intercept and slope parameters of MLR and BLR respectively.

I have found a tutorial on the internet about using a library called pymc3 to draw the fit line of BLR, I think his approach should be more correct, because it contains the same figure as in HW3.pdf, but I always encountered errors when I used pymc3, so I ended up using my own parameters to draw those fit line.

This is the URL of the teaching article:
https://github.com/WillKoehrsen/Data-Analysis/blob/master/bayesian_lr/Bayesian%20Linear%20Regression%20Demonstration.ipynb

And this is my figure:



It contains the two fit lines of MLR and BLR, and their intercepts and slopes are

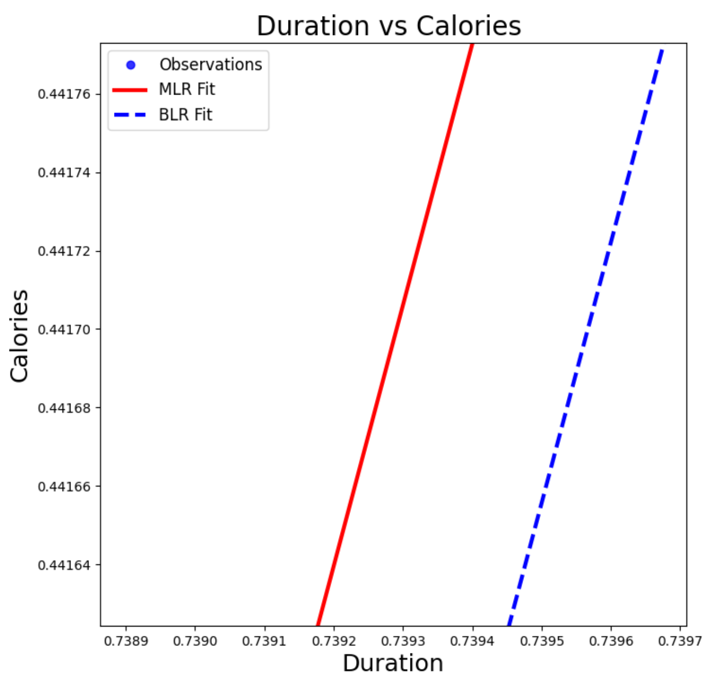
```
MLR Intercept: -0.05010506407846786
MLR Slope:  0.6652388496273769
BLR Intercept: -0.04970090101310507
BLR Slope:  0.6644441244717096
```

Calculate by

```
def MLR_coefficients(X, y):
    #co_MLR = np.matmul(np.matmul(np.linalg.inv(np.matmul(X.T, X)), X.T), y)
    coefs_MLR = np.linalg.inv(X.T @ X) @ X.T @ y                # calculate coffs of MLR, that is Intercept and Slope
    return coefs_MLR

def BLR_coefficients(X, y):
    coefs_BLR = np.linalg.inv(np.identity(X.shape[1]) + X.T @ X) @ X.T @ y          # calculate coffs of BLR, that is Intercept and Slope
    return coefs_BLR
```

The difference between the two is only a λI, so it seems to be overlapping, but when zoom in, can see that it is not.



4) (10 points) Please implement any regression model you want to get the best possible MSE. *use of built-in libraries is allowed only for this question.

In this part, I use LinearRegression model from sklearn, first call the model, and then train the model with train_data to make predictions

```
lr = LinearRegression()                                                          # use LinearRegression model from sklearn
lr.fit(train_data, train_label)                                                  # fit model
pred = lr.predict(test_data)                                                     # predict test_data using LinearRegression model
mse_lr_test = 1/len(test_data) * np.sum((test_label - pred)**2)                  # calculate mse of LinearRegression model
pred = lr.predict(validation_data)                                               # predict validation_data using LinearRegression model
mse_lr_validation= 1/len(validation_data) * np.sum((validation_label - pred)**2) # calculate mse of LinearRegression model
print('mse_lr_validation: ', mse_lr_validation, '\nmse_lr_test: ', mse_lr_test)
```

RESULT:

```
mse_lr_validation:  0.001329320794650206
mse_lr_test:  0.0012342166540746237
```

As can see that it is better than the MLR and BLR predictions I wrote myself.