Z5305320
Dong AO
(a):

$$\sum_{m=1}^{M}\sum_{i=1}^{|D_m|}\left(x_i^T\widehat{\beta_m}-y_i\right)^2, (x_i,y_i)\in D_m.$$

Note: $|D_m|$ means the size of D_m

(b):

1. If $\widehat{\beta_1}\dots\widehat{\beta_m}$ are fixed: For any data point (x_t,y_t) in D. We calculate the m values: $x_i^T\widehat{\beta_m}-y_i$ $(1\le m\le M)$ and we find the m which makes the least $x_i^T\widehat{\beta_m}-y_i$. We assign the point to D_m.
2. If $D_1\dots D_M$ are fixed: For every data point $(x_i,y_i)\in D_m$ $(1\le m\le M)$, we will try to minimize $\sum_{i=1}^{|D_m|}\left(x_i^T\widehat{\beta_m}-y_i\right)^2$, which is to learn $\widehat{\beta_m}$ here.
3. Return to process 1 and repeat until betas are not changed any more compared to last iteration.

(c)

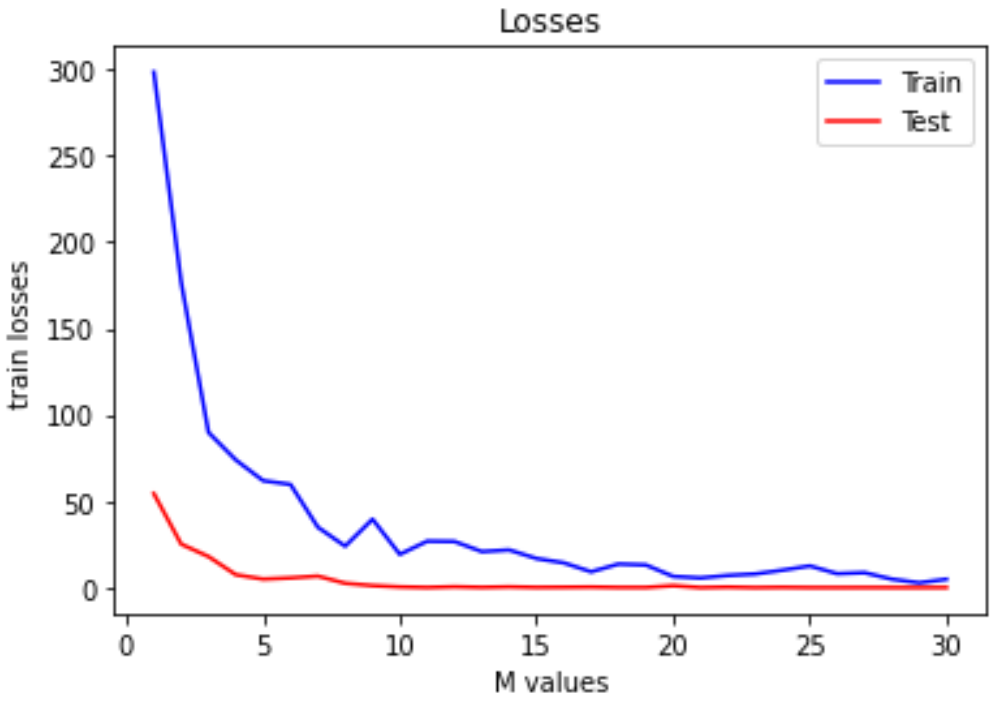
```
def total_loss(X, y, Z, models):
    '''
    computes total loss achieved on X, y based on linear regressions stored in models, and partit

    :param X: design matrix, n x p (np.array shape (n,p))
    :param y: response vector, n x 1 (np.array shape (n,1) or (n,))
    :param Z: assignment vector, n x 1 (assigns each sample to a partition)
    :param models: a list of M sklearn LinearRegression models, one for each of the partitions

    :returns: the loss of your complete model as computed in (a)
    '''

    loss = 0
    M = len(models)
    n = len(Z)

    for m in range(0, M):

        data_x_m = []
        data_y_m = []
        for i in range(n):
            if (Z[i] == m):
                data_x_m.append(X[i])
                data_y_m.append(y[i])

        if (len(data_x_m) > 0):
            y_pre = models[m].predict(data_x_m)

            one_loss = 0
            for i in range(len(y_pre)):
                if (Z[i] == m):
                    one_loss += (y_pre[i] - data_y_m[i])**2
            loss += one_loss

    return loss
```

(d):

```
def find_partitions(X, y, models):
    '''
    given M models, assigns points in X to one of the M partitions

    :param X: design matrix, n x p (np.array shape (n,p))
    :param y: response vector, n x 1 (np.array shape (n,1) or (n,))
    :param models: a list of M sklearn LinearRegression models for each
    of the partitions

    :returns: Z, a np.array of shape (n,) assigning each of the points in X to one
    '''
    M = len(models)
    n = len(y)
    Z = []
    for i in range(n):

        min_predict_y = 9999999
        target_m = 0
        for m in range(M):
            pred_y = models[m].predict([X[i]])

            if (pred_y - y[i])**2 < (min_predict_y - y[i])**2:
                min_predict_y = pred_y
                target_m = m
        Z.append(target_m)
    return np.array(Z)
```

(e):

I choose 20 since from the plot we can see it is the only one that train loss is not increasing at that point and test loss is small enough.



M	Train loss	Test loss
5	61.86355743	4.917106440992632
10	19.21842497	0.3965816078482734
15	16.82499839	0.128809950913761
20	6.47424378	1.1235300096590044
25	12.5374972	0.0096851771840851
30	4.83253453	0.0411392935295462

```
def get_new_models(x, y, z, models):
    M = len(models)
    n = len(y)
    for m in range(M):
        data_x_m = []
        data_y_m = []
        for i in range(n):
            if (Z[i] == m):
                data_x_m.append(x[i])
                data_y_m.append(y[i])
        if (len(data_x_m) > 0):
            models[m] = LinearRegression().fit(data_x_m, data_y_m)
    return models
```

```
n = 400

train_losses = []
test_losses = []
for m in range(1, 31):
    Z = np.array([i % m for i in range(n)])
    models = [LinearRegression().fit(x_train, y_train) for i in range(m)]
    models = get_new_models(x_train, y_train, Z, models)
    loss = total_loss(x_train, y_train, Z, models)
    loss_before = loss

    while (loss_before > loss):
        Z = find_partitions(x_train, y_train, models)
        models = get_new_models(x_train, y_train, Z, models)
        loss_before = loss
        loss = total_loss(x_train, y_train, Z, models)

    train_losses.append(loss)
    Z_test = find_partitions(x_test, y_test, models)
    test_losses.append(total_loss(x_test, y_test, Z_test, models)[0])
    print("M is equal to: ", m)
    print("loss train is: ", train_losses[-1])
    print("loss test is: ", test_losses[-1])
```

```
ax = plt.gca()
ax.plot(range(1,31), train_losses, color='b', label='Train')
ax.plot(range(1,31), test_losses, color='r', label='Test')
ax.legend()
plt.xlabel("M values")
plt.ylabel("train losses")
plt.title('Losses')
plt.show
```