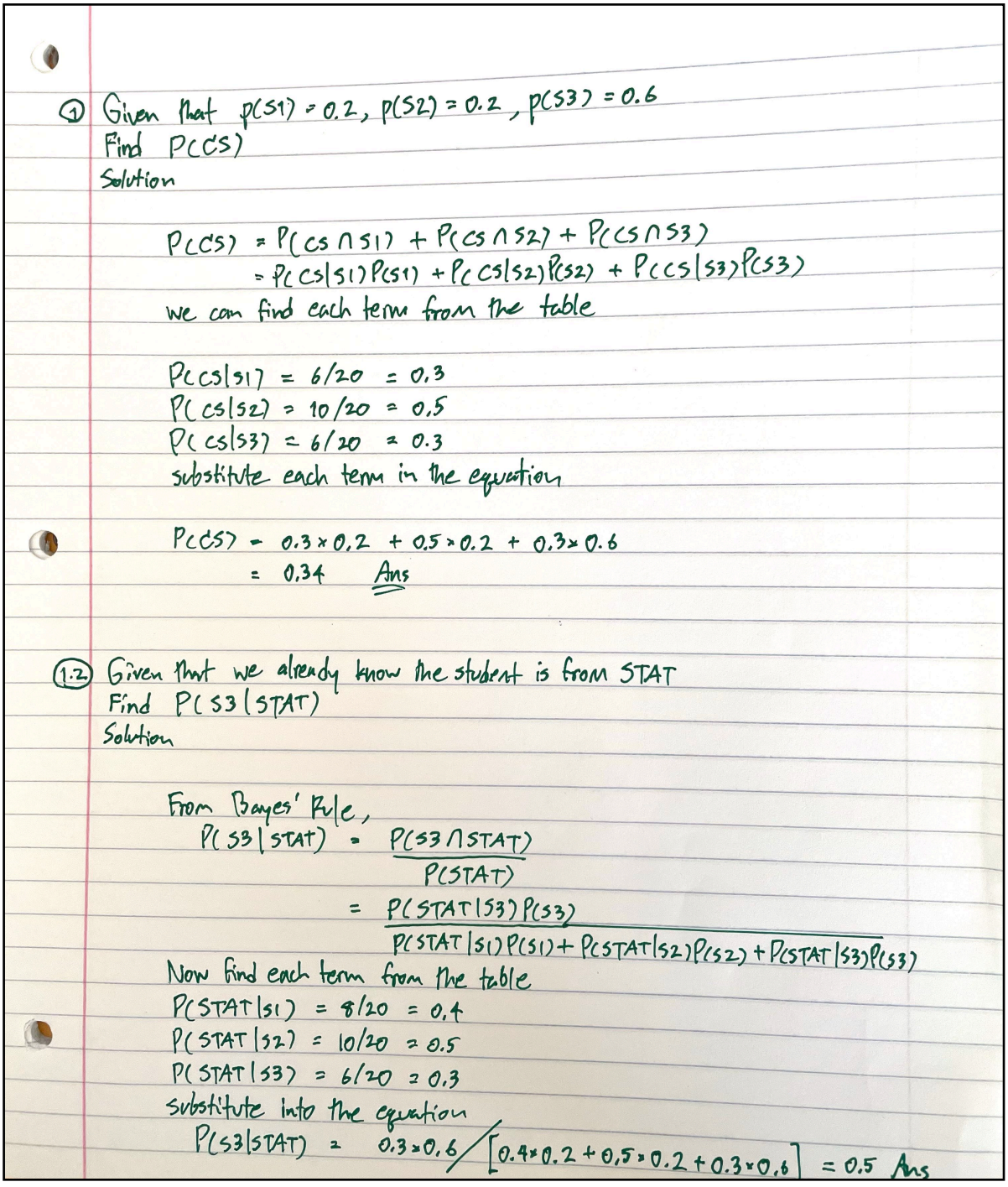
CS 559: Homework 1

Solution to Problems

1)



① Given that $p(S1) = 0.2$, $p(S2) = 0.2$, $p(S3) = 0.6$
Find $P(CS)$
Solution

$$P(CS) = P(CS \cap S1) + P(CS \cap S2) + P(CS \cap S3)$$
$$= P(CS|S1)P(S1) + P(CS|S2)P(S2) + P(CS|S3)P(S3)$$

we can find each term from the table

$P(CS|S1) = 6/20 = 0.3$
$P(CS|S2) = 10/20 = 0.5$
$P(CS|S3) = 6/20 = 0.3$
substitute each term in the equation

$$P(CS) = 0.3 \times 0.2 + 0.5 \times 0.2 + 0.3 \times 0.6$$
$$= 0.34 \quad \text{Ans}$$

1.2 Given that we already know the student is from STAT
Find $P(S3|STAT)$
Solution

From Bayes' Rule,

$$P(S3|STAT) = \frac{P(S3 \cap STAT)}{P(STAT)}$$
$$= \frac{P(STAT|S3)P(S3)}{P(STAT|S1)P(S1) + P(STAT|S2)P(S2) + P(STAT|S3)P(S3)}$$

Now find each term from the table
$P(STAT|S1) = 8/20 = 0.4$
$P(STAT|S2) = 10/20 = 0.5$
$P(STAT|S3) = 6/20 = 0.3$
substitute into the equation

$$P(S3|STAT) = 0.3 \times 0.6 / [0.4 \times 0.2 + 0.5 \times 0.2 + 0.3 \times 0.6] = 0.5 \quad \text{Ans}$$

2)

2.1)

Create a likelihood function based on the given samples.

```
samples = np.array([112,120,131,126,145,158,157,136,148,176])

mu = symbols('mu')
var = symbols('var')

def diff_sq(element,mean):
    result = np.square(element - mean)
    return result
vfunc = np.vectorize(diff_sq)
sum_term = np.sum(vfunc(samples,mu))

log_likelihood = (-1/(2*var))*sum_term - (samples.size/2)*log(var) - (samples.size/2)*np.log(2*np.pi)
log_likelihood
```

2.2)

Differentiate w.r.t mean

```
diff_likelihood_2_mu = diff(log_likelihood,mu)
diff_likelihood_2_mu
```

$$-\frac{20\mu - 2818}{2var}$$

Differentiate w.r.t. variance

```
diff_likelihiid_2_var = diff(log_likelihood,var)
diff_likelihiid_2_var
```

$$-\frac{5.0}{var} + \frac{(112-\mu)^2 + (120-\mu)^2 + (126-\mu)^2 + (131-\mu)^2 + (136-\mu)^2 + (145-\mu)^2 + (148-\mu)^2 + (157-\mu)^2 + (158-\mu)^2 + (176-\mu)^2}{2var^2}$$

Set the derivatives to 0 to obtain the local maximum

```
#set derivatives to 0 and solve 2 equations to get mu and var
sys = [diff_likelihood_2_mu,diff_likelihiid_2_var]
syms = [mu,var]

ans = nonlinsolve(sys,syms)
ans
```

$$\{(140.9,\ 0), (140.9,\ 346.69)\}$$

```
print(f"The maximum likelihood for mean and variance is {ans.args[0]}")
```

```
The maximum likelihood for mean and variance is (140.9, 346.69)
```

The values for mean and variance that maximize the likelihood are 140.9 and 346.69 respectively.

3)

3.1)

Create a likelihood function based on the probability mass function.

```
#Problem 3

q = symbols('q')
prob1 = 2*q/3
prob2 = q/3
prob3 = 2*(1-q)/3
prob4 = (1-q)/3

#find likelihood based on 10 independent observations
likelihood = (prob1**2)*(prob2**3)*(prob3**3)*(prob4**2)
likelihood
```

$$\frac{4q^5\left(\frac{1}{3}-\frac{q}{3}\right)^2\left(\frac{2}{3}-\frac{2q}{3}\right)^3}{243}$$

3.2)

Take the logarithm to the likelihood function.
Differentiate the log-likelihood function w.r.t. to q.
Set the derivative equation to 0 to find q that maximizes the likelihood.

```
log_likelood = log(likelihood)
log_likelood
```

$$\log\left(\frac{4q^5\left(\frac{1}{3}-\frac{q}{3}\right)^2\left(\frac{2}{3}-\frac{2q}{3}\right)^3}{243}\right)$$

```
diff_likelood_2_q = diff(log_likelood,q)
diff_likelood_2_q
```

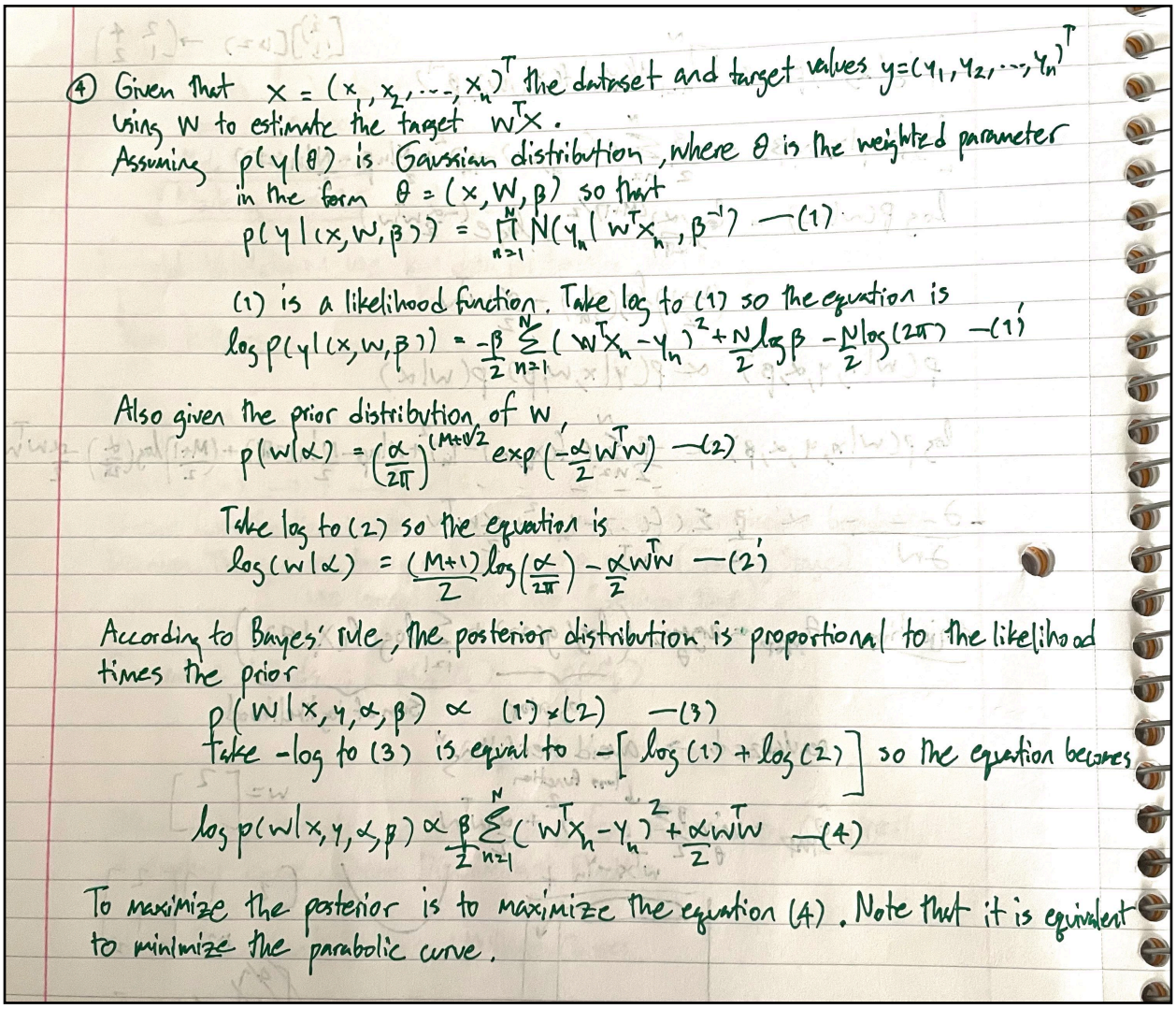
$$\frac{243\left(-\frac{8q^5\left(\frac{1}{3}-\frac{q}{3}\right)^2\left(\frac{2}{3}-\frac{2q}{3}\right)^2}{243}+\frac{4q^5\left(\frac{2}{3}-\frac{2q}{3}\right)^3\cdot\left(\frac{2q}{9}-\frac{2}{9}\right)}{243}+\frac{20q^4\left(\frac{1}{3}-\frac{q}{3}\right)^2\left(\frac{2}{3}-\frac{2q}{3}\right)^3}{243}\right)}{4q^5\left(\frac{1}{3}-\frac{q}{3}\right)^2\left(\frac{2}{3}-\frac{2q}{3}\right)^3}$$

```
ans2 = solve(diff_likelood_2_q,q)
print("maximum likelihood for q is ",ans2[0])
```

```
maximum likelihood for q is  1/2
```

Parameter q that gives the maximum likelihood is 0.5

4)



④ Given that $X = (x_1, x_2, \dots, x_n)^T$ the dataset and target values $y = (y_1, y_2, \dots, y_n)^T$
using w to estimate the target $w^T x$.
Assuming $p(y|\theta)$ is Gaussian distribution, where θ is the weighted parameter
in the form $\theta = (x, w, \beta)$ so that

$$p(y|(x,w,\beta)) = \prod_{n=1}^{N} N(y_n | w^T x_n, \beta^{-1}) \quad -(1)$$

(1) is a likelihood function. Take log to (1) so the equation is

$$\log p(y|(x,w,\beta)) = \frac{-\beta}{2} \sum_{n=1}^{N} (w^T x_n - y_n)^2 + \frac{N}{2} \log \beta - \frac{N}{2} \log(2\pi) \quad -(1')$$

Also given the prior distribution of w,

$$p(w|\alpha) = \left(\frac{\alpha}{2\pi}\right)^{(M+1)/2} \exp\left(-\frac{\alpha}{2} w^T w\right) \quad -(2)$$

Take log to (2) so the equation is

$$\log(w|\alpha) = \frac{(M+1)}{2} \log\left(\frac{\alpha}{2\pi}\right) - \frac{\alpha}{2} w^T w \quad -(2')$$

According to Bayes' rule, the posterior distribution is proportional to the likelihood
times the prior

$$p(w|x,y,\alpha,\beta) \propto (1) \times (2) \quad -(3)$$

Take -log to (3) is equal to $-[\log(1) + \log(2)]$ so the equation becomes

$$\log p(w|x,y,\alpha,\beta) \propto \frac{\beta}{2} \sum_{n=1}^{N} (w^T x_n - y_n)^2 + \frac{\alpha}{2} w^T w \quad -(4)$$

To maximize the posterior is to maximize the equation (4). Note that it is equivalent
to minimize the parabolic curve.

Note that equation (4) is a convex function that only has a global minimum, not a global maximum, and another term, $W^T * W$, regularizes the overall likelihood.

5)

After cleaning the data, it looks like the below figures

```
enb.head()
```

	X1	X2	X3	X4	X5	X6	X7	X8	Y2
0	0.98	514.5	294.0	110.25	7.0	2.0	0.0	0.0	21.33
1	0.98	514.5	294.0	110.25	7.0	3.0	0.0	0.0	21.33
2	0.98	514.5	294.0	110.25	7.0	4.0	0.0	0.0	21.33
3	0.98	514.5	294.0	110.25	7.0	5.0	0.0	0.0	21.33
4	0.90	563.5	318.5	122.50	7.0	2.0	0.0	0.0	28.28

```
enb.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   X1      768 non-null    float64
 1   X2      768 non-null    float64
 2   X3      768 non-null    float64
 3   X4      768 non-null    float64
 4   X5      768 non-null    float64
 5   X6      768 non-null    float64
 6   X7      768 non-null    float64
 7   X8      768 non-null    float64
 8   Y2      768 non-null    float64
dtypes: float64(9)
memory usage: 54.1 KB
```

The figures show that the cleaned data has 8 features and 768 entries as expected. Also, there is no ‘NULL’ entry in the dataset. Therefore, it is ready to be analyzed.

Use StandardScaler class from sklearn to normalize the dataset.

```
norm_enb = StandardScaler().fit_transform(enb)
norm_enb
```

```
array([[ 2.04177671, -1.78587489, -0.56195149, ..., -1.76044698,
        -1.81457514, -0.34266569],
       [ 2.04177671, -1.78587489, -0.56195149, ..., -1.76044698,
        -1.81457514, -0.34266569],
       [ 2.04177671, -1.78587489, -0.56195149, ..., -1.76044698,
        -1.81457514, -0.34266569],
       ...,
       [-1.36381225,  1.55394308,  1.12390297, ...,  1.2440492 ,
         1.41133622, -0.78654401],
       [-1.36381225,  1.55394308,  1.12390297, ...,  1.2440492 ,
         1.41133622, -0.83913623],
       [-1.36381225,  1.55394308,  1.12390297, ...,  1.2440492 ,
         1.41133622, -0.9001432 ]])
```

Create the cost function models, Lasso, Ridge, and Elastic, along with the gradients and linear regression.

```python
#Lasso Regression
def L1(X, Y, W, alpha=1):
    sum_w = 0
    for i in range(0,len(W)):
        sum_w = sum_w + abs(W[i])
    cost = np.dot((np.dot(X,W) - Y).T, np.dot(X,W) - Y)/len(Y) + alpha*sum_w
    return cost

#Ridge Regression
def L2(X, Y, W, alpha=1):
    sum_w = 0
    for i in range(0,len(W)):
        sum_w = sum_w + W[i]**2
    cost = np.dot((np.dot(X,W) - Y).T, np.dot(X,W) - Y)/len(Y) + alpha*sum_w
    return cost

#Elastic
def Elastic(X,Y,W,alpha):

    sum_w1 = 0
    sum_w2 = 0
    for i in range(0,len(W)):
        sum_w1 = sum_w1 + W[i]**2
        sum_w2 = sum_w2 + abs(W[i])
    cost = np.dot((np.dot(X,W) - Y).T, np.dot(X,W) - Y)/len(Y) + alpha*sum_w1 + (1-alpha)*sum_w2
    return cost
```

```python
#gradiant descent from Lasso
def gradient_L1(X,Y,W, alpha=1):
    s = np.zeros((X.shape[1],1))
    for i in range(0,len(Y)):
        for j in range(0,len(W)):
            s[j] += (-2/len(Y))*(Y[i] - np.dot(X[i,:],W))*X[i,j] + alpha*np.sign(W[j])
    return s

#gradiant descent from Ridge
def gradient_L2(X,Y,W, alpha=1):
    s = np.zeros((X.shape[1],1))
    for i in range(0,len(Y)):
        for j in range(0,len(W)):
            s[j] += (-2/len(Y))*(Y[i] - np.dot(X[i,:],W))*X[i,j] + 2*alpha*W[j]
    return s

#gradient descent from Elastic
def gradient_E(X,Y,W,alpha=0.5):
    s = np.zeros((X.shape[1],1))
    for i in range(0,len(Y)):
        for j in range(0,len(W)):
            s[j] += (-2/len(Y))*(Y[i] - np.dot(X[i,:],W))*X[i,j] + alpha*np.sign(W[j]) + (1-alpha)*2*W[j]
    return s
```

```python
def regression(X, Y, W, step, iterations, mode=1 ,alpha=1):

    cost_record = np.zeros((iterations,1))
    for iter in range(0,iterations):

        if mode == 1:
            s = gradient_L1(X,Y,W,alpha)
        elif mode == 2:
            s = gradient_L2(X,Y,W,alpha)
        else:
            s = gradient_E(X,Y,W,alpha)

        #update new W
        for k in range(0,len(W)):
            W[k] = W[k] - step*s[k]/len(Y)

        #update cost based on new W
        if mode == 1:
            cost_record[iter] = L1(X,Y,W,alpha)
        elif mode == 2:
            cost_record[iter] = L2(X,Y,W,alpha)
        else:
            cost_record[iter] = Elastic(X,Y,W,alpha)

        return W, cost_record
```

To test the models, create MSE validation and a predict function.

```
def predict(X,Y,W):
    Xb = np.c_[np.ones((len(Y),1)),X]
    return np.dot(Xb,W)

def mse(Y,Y_predict):
    n = len(Y)
    return round(((Y-Y_predict)**2).sum()/n,3)
```

Create the methods to train a model based on the regression function.

```
#train-test model
def train_L1(X,Y,step=0.01,alpha=0.2,iterations=100):

    W = np.ones((X.shape[1]+1,1))
    Xb = np.c_[np.ones((len(Y),1)),X]

    W_train, cost_history = regression(Xb,Y,W,step,iterations,alpha)

    return W_train, cost_history

def train_L2(X,Y,step=0.01,alpha=0.2,iterations=100):

    W = np.ones((X.shape[1]+1,1))
    Xb = np.c_[np.ones((len(Y),1)),X]

    mode = 2
    W_train, cost_history = regression(Xb,Y,W,step,iterations,mode,alpha)

    return W_train, cost_history

def train_Elastic(X,Y,step=0.01,alpha=0.2,iterations=100):

    W = np.ones((X.shape[1]+1,1))
    Xb = np.c_[np.ones((len(Y),1)),X]

    mode = 3
    W_train, cost_history = regression(Xb,Y,W,step,iterations,mode,alpha)

    return W_train, cost_history

def test_model(X,Y,W):

    Y_pred = predict(X,Y,W)
    return mse(Y,Y_pred)
```

Split the normalized data into train and test dataset based on the 5-fold validation rule. Use the train dataset to train 3 models 5 times, and the test dataset to test the 3 models. Collect MSE each time for the 3 models. After 5 times, calculate the average MSE and compare to obtain the least average MSE so that one model can be selected.

```
def k_validate(norm_data, number_run):

    model1 = np.zeros((number_run,1)) #L1 model
    model2 = np.zeros((number_run,1)) #L2 model
    model3 = np.zeros((number_run,1)) #Elastic model

    for k in range(0,number_run):
        X_train, X_test, Y_train, Y_test = train_test_split(X,Y, train_size = 0.9167, random_state=42)
        #train model
        #train_test_L1(X,Y,step=0.01,alpha=0.2,iterations=100)
        W_train1, cost_history1 = train_L1(X_train,Y_train)
        W_train2, cost_history2 = train_L2(X_train,Y_train)
        W_train3, cost_history3 = train_Elastic(X_train,Y_train)
        #test model
        model1[k] = test_model(X_test,Y_test,W_train1)
        model2[k] = test_model(X_test,Y_test,W_train2)
        model3[k] = test_model(X_test,Y_test,W_train3)

    #calculate avg
    avg1 = np.mean(model1, axis = 0)
    avg2 = np.mean(model2, axis = 0)
    avg3 = np.mean(model3, axis = 0)

    #print("avg1: ",avg1," ", "avg2: ",avg2," ","avg3: ",avg3)
    models = {"Lasso": avg1, "Ridge": avg2, "ElasticNet": avg3}
```

```
    minimum = 0

    if models["Lasso"] < models["Ridge"]:
        minimum = models["Lasso"]
        if minimum < models["ElasticNet"]:
            print("model ",get_keys_from_value(models,minimum)," has the best performance with average MSE of ",mini
        else:
            minimum = models["ElasticNet"]
            print("model ElasticNet has the best performance with average MSE of ",minimum)
    else:
        minimum = models["Ridge"]
        if minimum < models["ElasticNet"]:
            print("model ",get_keys_from_value(models,minimum)," has the best performance with average MSE of ",mini
        else:
            minimum = models["ElasticNet"]
            print("model ElasticNet has the best performance with average MSE of ",minimum)
```

```
#k_validate(norm_enb, 5) # 5-fold validation
k_validate(norm_enb,5)
```

```
model ElasticNet has the best performance with average MSE of  [4.345]
```