Kadir Alat

21502389

HW2

Secton : 2

Question 1.2.

1 . proportion of variance 438998.0770962068

2 . proportion of variance 409515.1272524627

3 . proportion of variance 356379.63526305044

Singular values directly affects the variances. Which also means singular values are almost same thing with eiugenvalues so its normal that singular values directly affects the variances.

Question 1 Code

```
import numpy as np
from matplotlib.image import imread
import os
from matplotlib import pyplot as plt

print(os.getcwd())
print(os.chdir("/Users/cumak/PycharmProjects/untitled/hw2/Van_gogh"))

all_pictures= []
names=[]

for file in os.listdir():
    names.append(file)
```

```
    temp = imread(file)
    all_pictures.append(temp)

#print(names)

grayscale = [128,128,128]
grayscale = np.array(grayscale)

"""for images in all_pictures:
    if(images.ndim==2):
        np.concatenate()"""
x=0



for images in all_pictures:
   if images.ndim == 2:
       temp= np.append(images,images)
       temp = np.append(temp, images)
       all_pictures[x]=temp.reshape(64,64,3)
       print(x)
       print(names[x])
       print(images.shape)
   x = x+1


noisy_dataset = np.array(all_pictures)
```

```
"""data2 = Image.fromarray(all_pictures[35])
data2.save("deneme3.jpg")"""

temp=0
array_x = []
for x in all_pictures:
    all_pictures[temp] = x.reshape(4096,3)
    array_x.append(all_pictures[temp])
    temp = temp + 1

array_x = np.array(array_x)
print(array_x.shape)

print(array_x[0].shape)
all_pictures = np.array(all_pictures)

print("boook",all_pictures.shape)

for i in range (3):
    U,S,VT = np.linalg.svd(all_pictures[:,:,i],full_matrices=False)
    S = np.diag(S)
    #print(S)
    temp = []
    temp2=[]
    a=0
    for x in range (877):
        for y in range(877):
            if(x==y):
                temp.append(S[x][y])
```

```
            if (a >= 10):
                pass
            else:
                temp2.append(S[x][y])
        a = a+1
        if(a==100):
          break

    temp = np.array(temp,dtype=float)
    #print(temp)
    f=[]
    for x in range(1,101):
        f.append(x)
    plt.bar(f,temp)
    plt.show()
    #print(i)
    print(i+1,". proportion of variance",sum(temp2))



#Question 1.2



def noisy(image):
      row,col,ch= image.shape
      mean = np.mean(image)
      var = np.var(image)
      sigma = var**0.5
      gauss = np.random.normal(mean,sigma,(row,col,ch))
      gauss = gauss.reshape(row,col,ch)
```

```
    noisy = image + 0.01*gauss
    return noisy

images_with_noise=[]

for i in noisy_dataset:
  images_with_noise.append(noisy(i))



temp=0
for x in images_with_noise:
  images_with_noise[temp] =images_with_noise[temp].reshape(4096,3)
  temp = temp + 1

images_with_noise = np.array(images_with_noise)
print(images_with_noise.shape)



for i in range (3):
  U,S,VT = np.linalg.svd(images_with_noise[:,:,i],full_matrices=False)
  S = np.diag(S)
  #print(S)
  temp = []
  temp2=[]
  a=0
  for x in range (877):
    for y in range(877):
      if(x==y):
        temp.append(S[x][y])
```

```
            if (a >= 10):
                pass
            else:
                temp2.append(S[x][y])
        a = a+1
        if(a==100):
          break
    temp = np.array(temp,dtype=float)
    print(i+1,". proportion of variance with noise",sum(temp2))


#first = noisy(all_pictures[0])
#print(first)
```

Question 2.1

$J_n = ||y - X\beta||^2 = (y - X\beta)^T * (y - X\beta)$

We can represent loss function like (dimension of matrix is the example is at random):

$$\frac{1}{2}\sum_{i=1}^{m}(yprecidted(i) - y(i))^2 \quad => \quad \begin{bmatrix} x10 & \cdots & x12 \\ \vdots & \ddots & \vdots \\ xm0 & \cdots & xm2 \end{bmatrix} * \begin{bmatrix} \beta 0 \\ . \\ . \\ \beta m \end{bmatrix} = \begin{bmatrix} \text{ypredicted1} \\ . \\ . \\ ypredictedm \end{bmatrix}$$

$(ypredicted-y)^2$ will have a scalar result. That's why we can express the square notation by multiplying transpose of matrix with the matrix so,

$J_n = (y-X\beta)^2 = (y-X\beta)^T * (y-X\beta)$

$J_n = (y^T-\beta^TX^T)*(y-X\beta)$

$J_n = y^Ty - y^TX\beta - y\beta^TX^T + \beta^TX^T X\beta$

Take first derivative of β :

$\nabla_{\beta}(y^{T}y - y^{T}X\beta - y\beta^{T}X^{T} + \beta^{T}X^{T}X\beta)$

$= 2X^{T}X\beta - \nabla_{\beta}(y^{T}X\beta + y\beta^{T}X^{T})$

$=2X^{T}X\beta - 2X^{T}Y = 0$ so,

, $\beta = (X^{T}X)^{-1}X^{T}Y$ // that equation is used in coding

Question 2.2

The term "fold" represented as like:

1st fold means :

First 100 examples are stored for testing others for training,

2nd fold means:

Examples between 100-200 are stored for testing others for training etc.

Results:

1st fold

R squared result for first fold 0.678795416364816

Mean square Error Result for first fold 0.00980581321618013

Mean absolute Error Result for first fold 0.07056263713048568

MAPE Error Result for first fold 0.13338997248956544

2nd fold

R squared result for second fold 0.9154187925706662

Mean square Error Result for second fold 0.0030901879532006914

Mean absolute Error Result for second fold 0.04360884271662547

MAPE Error Result for second fold 0.06135227749217452

3rd fold

R squared result for third fold 1.165099167176053

Mean square Error Result for third fold 0.0016945864071199386

Mean absolute Error Result for third fold 0.034059869297647594

MAPE Error Result for third fold 0.04742626526722578

4th fold

R squared result for fourth fold 0.8994013190288073

Mean square Error Result for fourth fold 0.0034383213553692477

Mean absolute Error Result for fourth fold 0.04235448795123107

MAPE Error Result for fourth fold 0.06826057372602094

5th fold

R squared result for fifth fold 0.8856686105755853

Mean square Error Result for fifth fold 0.0018431744376048633

Mean absolute Error Result for fifth fold 0.03390995704418458

MAPE Error Result for fifth fold 0.052007762035798544

Question 2.3

Coordinate Descent for least squares regression:

$z_j = \sum_{i+1}^{N} h_j(x_i))^2$ //h_j is the normalization function for the dataset

$p_j = \sum_{i=0}^{N} h_j(x_i)(y_i - \hat{y}_i(\hat{w}_{-j}))$

then set the weights in regard of correlation between p_j and the λ that we gave the value of it.

R squared result for first fold_lasso (FIRST FOLD) 0.5732031998813681

Mean square Error Result for first fold_lasso (FIRST FOLD) 0.0076512842585096715

Mean absolute Error Result for first fold_lasso (FIRST FOLD) 0.07544802476774977

MAPE Error Result for first fold _lasso (FIRST FOLD) 0.12473487236185601

R squared result for second fold_lasso (SECOND FOLD) 1.0988496174603124

Mean square Error Result for second fold_lasso (SECOND FOLD) 0.007071783048793769

Mean absolute Error Result for second fold_lasso (SECOND FOLD) 0.07465021327784348

MAPE Error Result for second fold _lasso (SECOND FOLD) 0.09754879699480519

R squared result for third fold_lasso (THIRD FOLD) 1.3407144796634791

Mean square Error Result for third fold_lasso (THIRD FOLD) 0.005642182165136504

Mean absolute Error Result for third fold_lasso (THIRD FOLD) 0.06817830141129337

MAPE Error Result for third fold _lasso (THIRD FOLD) 0.09194003515369856

R squared result for fourth fold_lasso (FOURTH FOLD) 0.9695973877652866

Mean square Error Result for fourth fold_lasso (FOURTH FOLD) 0.006921838837341102

Mean absolute Error Result for fourth fold_lasso (FOURTH FOLD) 0.0727700526004623

MAPE Error Result for fourth fold _lasso (FOURTH FOLD) 0.10706291791559754

R squared result for fifth fold_lasso (FIFTH FOLD) 0.9517111414422369

Mean square Error Result for fifth fold_lasso (FIFTH FOLD) 0.004820154317713675

Mean absolute Error Result for fifth fold_lasso (FIFTH FOLD) 0.06120120865762344

MAPE Error Result for fifth fold _lasso (FIFTH FOLD) 0.08606257762202801

Question 2 Code

```
import csv
import numpy as np
from matplotlib import pyplot as plt
```

```
independent_features = []
temp_independent_list=[]
dependent_features=[]
with open('q2_dataset.csv','r') as csv_file:
    reader = csv.reader(csv_file)
    a =1
    next(reader)
    for line in reader:
        for value in line:
            if (a == len(line)):
                pass
            else:
                temp_independent_list.append(value)
            if (a==len(line)-1):
                temp_independent_list.append(1)      #adding ones to end of independent feature
rows
            if(a==len(line)):
                dependent_features.append(value)
            a = a + 1
        independent_features.append(temp_independent_list)
        temp_independent_list=[]
        a=1



def get_weights(train_set,train_label):
    transpose = np.transpose(train_set)
    multiplication = np.matmul(transpose,train_set)
    inverse = np.linalg.inv(multiplication)
    before_theta = np.matmul(inverse,transpose)
    theta = np.matmul(before_theta,train_label)
```

```
    return theta


def chose_test_fold(num,independent_features,dependent_features): # number between 1-5 will determine the partition that will use for testing others for training


    test_fold = []
    test_fold_label=[]
    train_fold = []
    train_fold_label = []
    if (num == 1):
        i=0
        while(i<100):
            test_fold.append(independent_features[i])
            test_fold_label.append(dependent_features[i])
            i += 1
        while(i<500):
            train_fold.append(independent_features[i])
            train_fold_label.append(dependent_features[i])
            i +=1
    elif (num == 2):
        i=0
        while(i<100):
            train_fold.append( independent_features[i])
            train_fold_label.append(dependent_features[i])
            i += 1
        while (i <200):
            test_fold.append(independent_features[i])
            test_fold_label.append(dependent_features[i])
            i += 1
        while(i<500):
```

```
        train_fold.append(independent_features[i])
        train_fold_label.append(dependent_features[i])
        i +=1

elif (num == 3):
    i=0
    while(i<200):
      train_fold.append( independent_features[i])
      train_fold_label.append(dependent_features[i])
      i += 1
    while (i < 300):
        test_fold.append(independent_features[i])
        test_fold_label.append(dependent_features[i])
        i += 1
    while(i<500):
        train_fold.append(independent_features[i])
        train_fold_label.append(dependent_features[i])
        i +=1

elif (num == 4):
    i=0
    while(i<300):
      train_fold.append(independent_features[i])
      train_fold_label.append(dependent_features[i])
      i += 1
    while (i < 400):
        test_fold.append(independent_features[i])
        test_fold_label.append(dependent_features[i])
        i += 1
```

```
        while(i<500):
            train_fold.append(independent_features[i])
            train_fold_label.append(dependent_features[i])
            i +=1

    elif (num == 5):
        i=0
        while(i<400):
            train_fold.append( independent_features[i])
            train_fold_label.append(dependent_features[i])
            i += 1
        while(i<500):
            test_fold.append(independent_features[i])
            test_fold_label.append(dependent_features[i])
            i +=1

    train_fold=np.array(train_fold,dtype=float)
    train_fold_label=np.array(train_fold_label,dtype=float)
    test_fold=np.array(test_fold,dtype=float)
    test_fold_label=np.array(test_fold_label,dtype=float)

    return (train_fold,train_fold_label,test_fold,test_fold_label)

def r_squared_error(real,estimation):
    mean = sum(real)/len(real)
    denominator_array = real-mean
    denominator =0
    numerator_array = estimation-mean
    for i1 in denominator_array:
```

```
        denominator += i1*i1
    numerator=0
    for i2 in numerator_array:
        numerator += i2*i2
    result = numerator/denominator
    return result
def mean_squared_error(real,estimation):
    x=real-estimation
    total = 0
    for i in x:
        total += i*i
    result = total / len(x)
    return result
def mean_absolute_error(real,estimation):
    x = real-estimation
    total = 0
    for i in x:
        if(i<0):
            i = abs(i)
            total += i
        else:
            total +=i
    result = total/len(x)
    return result
def mape_error(real,estimation):
    x = real-estimation
    temp = []
    a=0
    for i in x:
```

```
        temp.append(abs(i/real[a]))
        a += 1
    total = 0
    for a in temp:
        total += a
    result = total/len(x)
    return result
def lasso_regularization(theta):
    lamda = 1
    theta_penalty = 0
    for i in theta:
        theta_penalty += lamda*abs(i)
    theta = theta + theta_penalty
    return theta



folds_dictionary = {}
for i in range(1,6):
    folds_dictionary[i] = chose_test_fold(i,independent_features,dependent_features)
#print(folds_dictionary)

# first fold is test
theta_first = get_weights(folds_dictionary[1][0],folds_dictionary[1][1])
est1 = np.dot(folds_dictionary[1][2],np.transpose(theta_first))
real1 = folds_dictionary[1][3]
r_sq1 = r_squared_error(real1,est1)
print("R squared result for first fold",r_sq1)
mse1=mean_squared_error(real1,est1)
print("Mean square Error Result for first fold",mse1)
```

```
mae1=mean_absolute_error(real1,est1)
print("Mean absolute Error Result for first fold",mae1)
mape1=mape_error(real1,est1)
print("MAPE Error Result for first fold",mape1)


# second fold is test
theta_second = get_weights(folds_dictionary[2][0],folds_dictionary[2][1])
est2 = np.dot(folds_dictionary[2][2],np.transpose(theta_second))
real2 = folds_dictionary[2][3]
r_sq2 = r_squared_error(real2,est2)
print("R squared result for second fold",r_sq2)
mse2=mean_squared_error(real2,est2)
print("Mean square Error Result for second fold",mse2)
mae2=mean_absolute_error(real2,est2)
print("Mean absolute Error Result for second fold",mae2)
mape2=mape_error(real2,est2)
print("MAPE Error Result for second fold",mape2)


# third fold is test
theta_third = get_weights(folds_dictionary[3][0],folds_dictionary[3][1])
est3 = np.dot(folds_dictionary[3][2],np.transpose(theta_third))
real3 = folds_dictionary[3][3]
r_sq3 = r_squared_error(real3,est3)
print("R squared result for third fold",r_sq3)
mse3=mean_squared_error(real3,est3)
print("Mean square Error Result for third fold",mse3)
mae3=mean_absolute_error(real3,est3)
```

```
print("Mean absolute Error Result for third fold",mae3)
mape3=mape_error(real3,est3)
print("MAPE Error Result for third fold",mape3)



# fourth fold is test
theta_fourth = get_weights(folds_dictionary[4][0],folds_dictionary[4][1])
est4 = np.dot(folds_dictionary[4][2],np.transpose(theta_fourth))
real4 = folds_dictionary[4][3]
r_sq4 = r_squared_error(real4,est4)
print("R squared result for fourth fold",r_sq4)
mse4=mean_squared_error(real4,est4)
print("Mean square Error Result for fourth fold",mse4)
mae4=mean_absolute_error(real4,est4)
print("Mean absolute Error Result for fourth fold",mae4)
mape4=mape_error(real4,est4)
print("MAPE Error Result for fourth fold",mape4)



# fifth fold is test
theta_fifth = get_weights(folds_dictionary[5][0],folds_dictionary[5][1])
est5 = np.dot(folds_dictionary[5][2],np.transpose(theta_fifth))
real5 = folds_dictionary[5][3]
r_sq5 = r_squared_error(real5,est5)
print("R squared result for fifth fold",r_sq5)
mse5=mean_squared_error(real5,est5)
print("Mean square Error Result for fifth fold",mse5)
```

```
mae5=mean_absolute_error(real5,est5)
print("Mean absolute Error Result for fifth fold",mae5)
mape5=mape_error(real5,est5)
print("MAPE Error Result for fifth fold",mape5)

#Question 3.2

feature_set_independent=[]
for x1 in folds_dictionary[1][2]:
    feature_set_independent.append(x1)
    #print(type(x1))
for x2 in folds_dictionary[1][0]:
    feature_set_independent.append(x2)
feature_set_independent=np.array(feature_set_independent)

feature_set_dependent=[]
for x1 in folds_dictionary[1][3]:
    feature_set_dependent.append(x1)
for x2 in folds_dictionary[1][1]:
    feature_set_dependent.append(x2)
feature_set_dependent=np.array(feature_set_dependent)



#normalization of feature set
columns = []
columns_temp=[]
i=0
for i in range(len(feature_set_independent[1])):
    columns.append(feature_set_independent[:,i])
```

```
columns = np.array(columns)
sum_of_columns = []
for i in columns:
  sum_of_columns.append(sum(abs(i)))
for i in range(len(columns)):
   columns[i] = columns[i]/sum_of_columns[i]



#print(theta_first)
estimation = np.matmul(theta_first.transpose(),columns)
#print("Theta firsy",theta_first)
#print("columns",columns)
#print("estimation",estimation)
temp=[]
for i in range(500):
 temp.append(round(estimation[i] - feature_set_dependent[i],2))
estimation2 = temp
estimation2 = np.array(estimation2)
#print(estimation2)
H = np.matmul(2*columns,estimation2)
#print("H değerleri",H)
for i in range(len(H)):
   H[i] = H[i]/sum(H)
#print("H değerleri",H)

lmd=0.01
new_weights=[]
#print("Theta first",theta_first)
```

```
for i in range(len(H)):
    if(theta_first[i]>=0):
        new_weights.append(theta_first[i]-0.001*(H[i]+lmd))
    else:
        new_weights.append((theta_first[i]-0.001*(H[i]-lmd)))
#print("New weights",new_weights)


est1_lasso = np.dot(folds_dictionary[1][2],np.transpose(new_weights))
real1_lasso = folds_dictionary[1][3]
r_sq1_lasso = r_squared_error(real1_lasso,est1_lasso)
print("R squared result for first fold_lasso (FIRST FOLD)",r_sq1_lasso)
mse1_lasso=mean_squared_error(real1_lasso,est1_lasso)
print("Mean square Error Result for first fold_lasso (FIRST FOLD)",mse1_lasso)
mae1_lasso=mean_absolute_error(real1_lasso,est1_lasso)
print("Mean absolute Error Result for first fold_lasso (FIRST FOLD)",mae1_lasso)
mape1_lasso=mape_error(real1_lasso,est1_lasso)
print("MAPE Error Result for first fold _lasso (FIRST FOLD) ",mape1_lasso)

est2_lasso = np.dot(folds_dictionary[2][2],np.transpose(new_weights))
real2_lasso = folds_dictionary[2][3]
r_sq2_lasso = r_squared_error(real2_lasso,est2_lasso)
print("R squared result for second fold_lasso (SECOND FOLD)",r_sq2_lasso)
mse2_lasso=mean_squared_error(real2_lasso,est2_lasso)
print("Mean square Error Result for second fold_lasso (SECOND FOLD)",mse2_lasso)
mae2_lasso=mean_absolute_error(real2_lasso,est2_lasso)
print("Mean absolute Error Result for second fold_lasso (SECOND FOLD)",mae2_lasso)
mape2_lasso=mape_error(real2_lasso,est2_lasso)
print("MAPE Error Result for second fold _lasso (SECOND FOLD)",mape2_lasso)
```

```
est3_lasso = np.dot(folds_dictionary[3][2],np.transpose(new_weights))
real3_lasso = folds_dictionary[3][3]
r_sq3_lasso = r_squared_error(real3_lasso,est3_lasso)
print("R squared result for third fold_lasso (THIRD FOLD)",r_sq3_lasso)
mse3_lasso=mean_squared_error(real3_lasso,est3_lasso)
print("Mean square Error Result for third fold_lasso (THIRD FOLD)",mse3_lasso)
mae3_lasso=mean_absolute_error(real3_lasso,est3_lasso)
print("Mean absolute Error Result for third fold_lasso (THIRD FOLD)",mae3_lasso)
mape3_lasso=mape_error(real3_lasso,est3_lasso)
print("MAPE Error Result for third fold _lasso (THIRD FOLD)",mape3_lasso)

est4_lasso = np.dot(folds_dictionary[4][2],np.transpose(new_weights))
real4_lasso = folds_dictionary[4][3]
r_sq4_lasso = r_squared_error(real4_lasso,est1_lasso)
print("R squared result for fourth fold_lasso (FOURTH FOLD)",r_sq4_lasso)
mse4_lasso=mean_squared_error(real4_lasso,est4_lasso)
print("Mean square Error Result for fourth fold_lasso (FOURTH FOLD)",mse4_lasso)
mae4_lasso=mean_absolute_error(real4_lasso,est4_lasso)
print("Mean absolute Error Result for fourth fold_lasso (FOURTH FOLD)",mae4_lasso)
mape4_lasso=mape_error(real4_lasso,est4_lasso)
print("MAPE Error Result for fourth fold _lasso (FOURTH FOLD)",mape4_lasso)

est5_lasso = np.dot(folds_dictionary[5][2],np.transpose(new_weights))
real5_lasso = folds_dictionary[5][3]
r_sq5_lasso = r_squared_error(real5_lasso,est5_lasso)
print("R squared result for fifth fold_lasso (FIFTH FOLD)",r_sq5_lasso)
mse5_lasso=mean_squared_error(real5_lasso,est5_lasso)
print("Mean square Error Result for fifth fold_lasso (FIFTH FOLD)",mse5_lasso)
```

```
mae5_lasso=mean_absolute_error(real5_lasso,est5_lasso)
print("Mean absolute Error Result for fifth fold_lasso (FIFTH FOLD)",mae5_lasso)
mape5_lasso=mape_error(real5_lasso,est5_lasso)
print("MAPE Error Result for fifth fold _lasso (FIFTH FOLD)",mape5_lasso)



f1=["R-SQ","R-SQ(L1)"]
temp1 = [r_sq1,r_sq1_lasso]
plt.bar(f1,temp1)
plt.title("Fold 1")
plt.show()


f2=["MSE","MSE(L1)"]
temp2 = [mse1,mse1_lasso]
plt.bar(f2,temp2)
plt.title("Fold 1")
plt.show()

f3=["MAE","MAE(L1)"]
temp3 = [mae1,mae1_lasso]
plt.bar(f3,temp3)
plt.title("Fold 1")
plt.show()

f4=["MAPE","MAPE(L1)"]
temp4 = [mape1,mape1_lasso]
```

```
plt.bar(f4,temp4)
plt.title("Fold 1")
plt.show()


f1=["R-SQ","R-SQ(L1)"]
temp1 = [r_sq2,r_sq2_lasso]
plt.bar(f1,temp1)
plt.title("Fold 2")
plt.show()


f2=["MSE","MSE(L1)"]
temp2 = [mse2,mse2_lasso]
plt.bar(f2,temp2)
plt.title("Fold 2")
plt.show()

f3=["MAE","MAE(L1)"]
temp3 = [mae2,mae2_lasso]
plt.bar(f3,temp3)
plt.title("Fold 2")
plt.show()

f4=["MAPE","MAPE(L1)"]
temp4 = [mape2,mape2_lasso]
plt.bar(f4,temp4)
plt.title("Fold 2")
plt.show()
```

```
f1=["R-SQ","R-SQ(L1)"]
temp1 = [r_sq3,r_sq3_lasso]
plt.bar(f1,temp1)
plt.title("Fold 3")
plt.show()


f2=["MSE","MSE(L1)"]
temp2 = [mse3,mse3_lasso]
plt.bar(f2,temp2)
plt.title("Fold 3")
plt.show()

f3=["MAE","MAE(L1)"]
temp3 = [mae3,mae3_lasso]
plt.bar(f3,temp3)
plt.title("Fold 3")
plt.show()

f4=["MAPE","MAPE(L1)"]
temp4 = [mape3,mape3_lasso]
plt.bar(f4,temp4)
plt.title("Fold 3")
plt.show()


f1=["R-SQ","R-SQ(L1)"]
```

```
temp1 = [r_sq4,r_sq4_lasso]
plt.bar(f1,temp1)
plt.title("Fold 4")
plt.show()


f2=["MSE","MSE(L1)"]
temp2 = [mse4,mse4_lasso]
plt.bar(f2,temp2)
plt.title("Fold 4")
plt.show()

f3=["MAE","MAE(L1)"]
temp3 = [mae4,mae4_lasso]
plt.bar(f3,temp3)
plt.title("Fold 4")
plt.show()

f4=["MAPE","MAPE(L1)"]
temp4 = [mape4,mape4_lasso]
plt.bar(f4,temp4)
plt.title("Fold 4")
plt.show()



f1=["R-SQ","R-SQ(L1)"]
temp1 = [r_sq5,r_sq5_lasso]
plt.bar(f1,temp1)
```

```
plt.title("Fold 5")
plt.show()


f2=["MSE","MSE(L1)"]
temp2 = [mse5,mse5_lasso]
plt.bar(f2,temp2)
plt.title("Fold 5")
plt.show()

f3=["MAE","MAE(L1)"]
temp3 = [mae5,mae5_lasso]
plt.bar(f3,temp3)
plt.title("Fold 5")
plt.show()

f4=["MAPE","MAPE(L1)"]
temp4 = [mape5,mape5_lasso]
plt.bar(f4,temp4)
plt.title("Fold 5")
plt.show()
```