CS464 Introduction to Machine Learning

Fall 2021

Homework 2

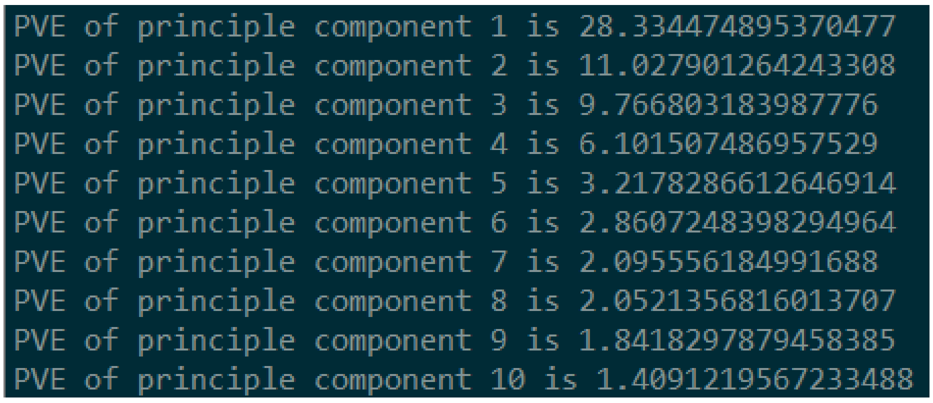
Ata Berk ÇAKIR

21703127

Question 1

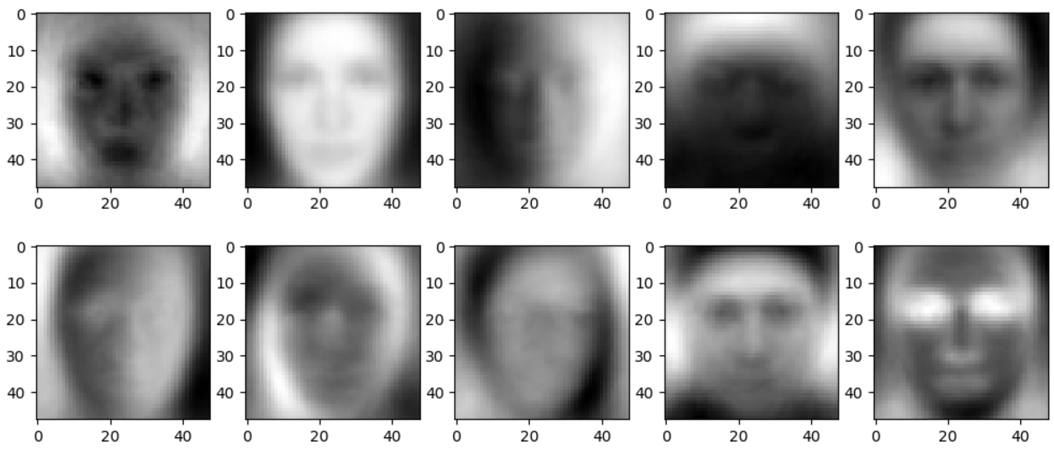
1.1

Proportion of variance explaned for the first 10 principle component is the following.



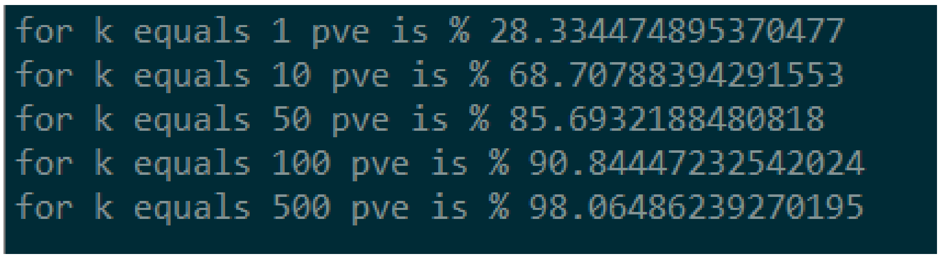
```
PVE of principle component 1 is 28.334474895370477
PVE of principle component 2 is 11.027901264243308
PVE of principle component 3 is 9.766803183987776
PVE of principle component 4 is 6.101507486957529
PVE of principle component 5 is 3.2178286612646914
PVE of principle component 6 is 2.8607248398294964
PVE of principle component 7 is 2.095556184991688
PVE of principle component 8 is 2.0521356816013707
PVE of principle component 9 is 1.8418297879458385
PVE of principle component 10 is 1.4091219567233488
```

After obtanin first 10 eigen vectors of first 10 principle components ı shaped each of the eigen vector into 48X48 form in order to plot them as eigenfaces. To do that i used the imshow() function of the pyplot library. The eigen faces corresponds to first 10 eigen vectors are the following. PCA (Principal Component Analysis) is a technique for reducing dimensionality. It reduces dimensionality and projects a training sample/data into a compact feature space using Eigenvalues and EigenVectors. As seen in the figure we can see the reduced versions of the faces as eigenfaces.



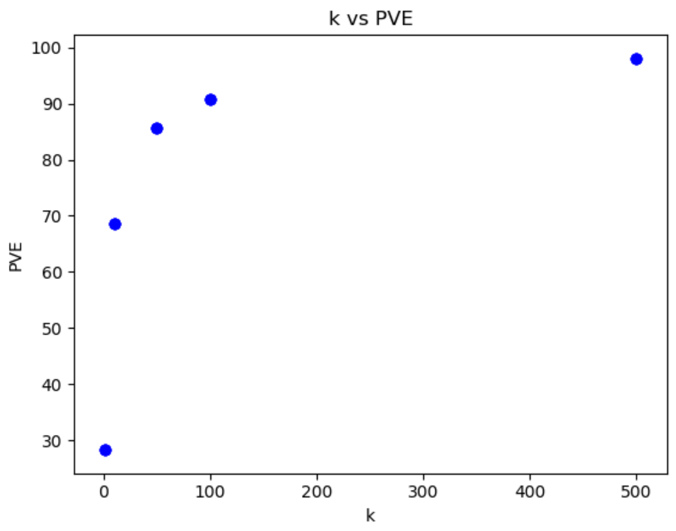
1.2

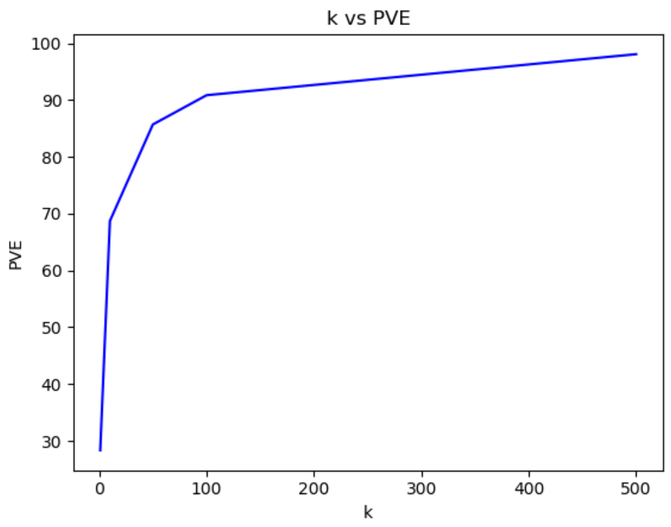
First k principal components PVE for k ∈ {1, 10, 50, 100, 500} is the following. For this calculation cumalative pve method is used ib other words all pves are summed for the corresponding k value.



```
for k equals 1 pve is % 28.334474895370477
for k equals 10 pve is % 68.70788394291553
for k equals 50 pve is % 85.6932188480818
for k equals 100 pve is % 90.84447232542024
for k equals 500 pve is % 98.06486239270195
```

K vs PVE plot is the following.





As seen in the plot when k values increses PVE values are becoming closer to %100.

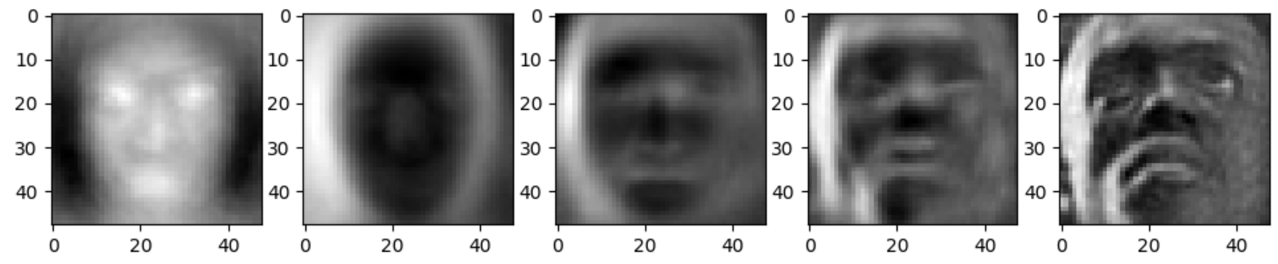
1.3

In order to reconstruct an image using the principal components obtained in question 1.1 can be done by the following formula.

$$\widehat{D} = DV^{T}V + mean$$

Where $\widehat{D}$ is the reconstructed image, D is the mean centered original data and V is the eigen vectors in original matrix form(2304x10).

First k principal components to analyze and reconstruct the first image in the dataset where k ∈ 1, 10, 50, 100, 500 is the folowing. As seen in the figure when k values are increases the image becomeing more clear as expect. The reason behind is PVE is increasing when k values are increasing which is shown by a plot in the question 1.2



Question 2

2.1

We are going to minimize the loss function (MSE) to find the weights. To do that we need to solve the equation given in the homework description where its gradient is equal to 0.

$$\nabla_\beta (X\beta - y)^T (X\beta - y) = 0$$

$$\nabla_\beta (\beta^T X^T X\beta - 2\beta^T X^T y + y^T y) = 0$$

$$2X^T X\beta - 2X^T y = 0$$

$$\beta = (X^T X)^{-1} X^T y$$

Where β is the weights X is the train features and y is the train labels. Hence the formula for linear regression is the following

$$\hat{y} = \beta^T X + b$$

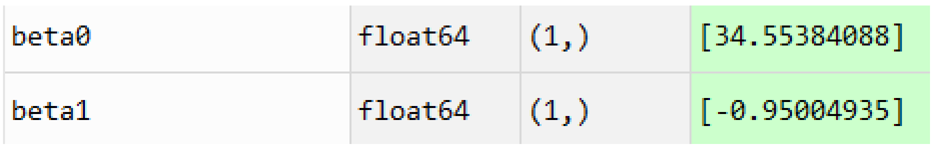
Where b is the bias or weight zero (β_0) in other words.

2.2

We know that a square matrix of full rank is invertible. $X^T X$ is 13 by 13 matrix and according to de calculations rank of it is equal to 13 this means that $X^T X$ is full rank. Therefore, we can assume that it is invertable.

2.3

According to the calculation done by the weight formula given above the resultant weight matrix is the following

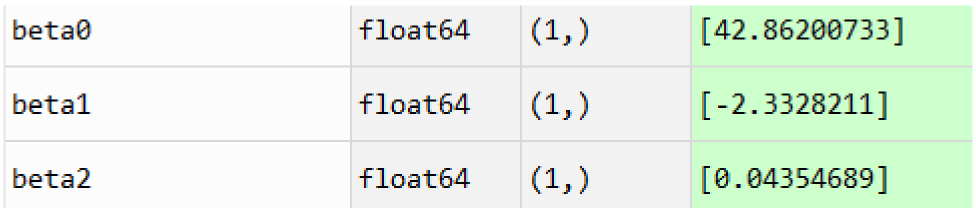


beta0	float64	(1,)	[34.55384088]
beta1	float64	(1,)	[-0.95004935]

The plot will be provided in the next question together with the results of the polynomial regression for the easiness of the comment.

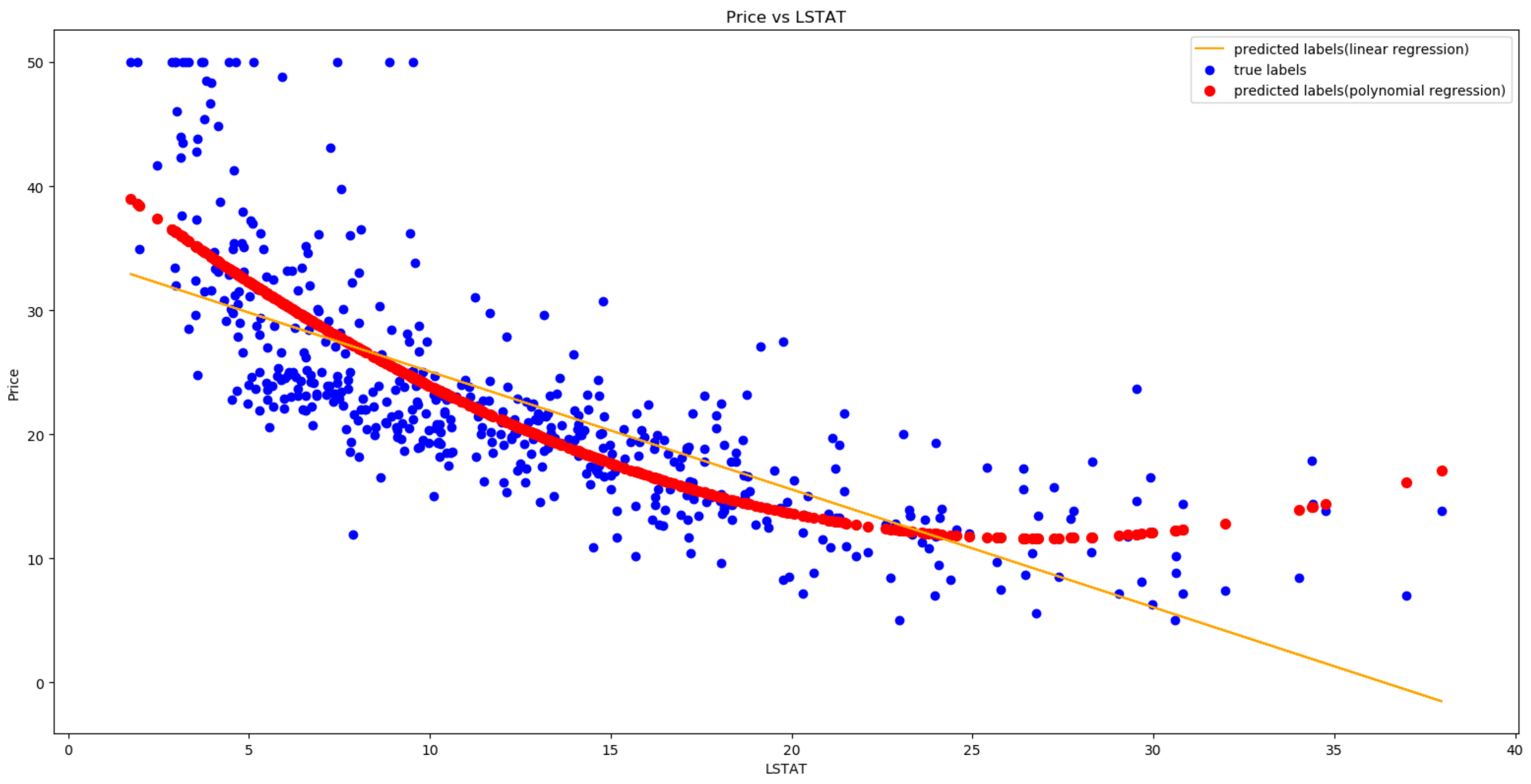
2.4

According to the calculation done by the weight formula given above the resultant weight matrix is the following

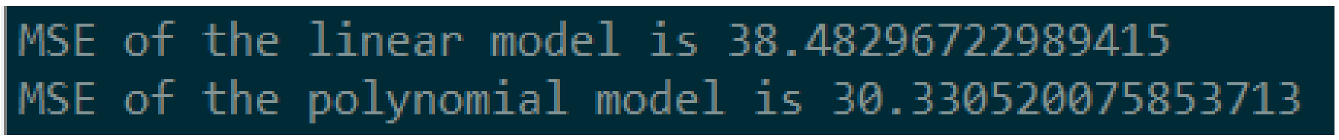


beta0	float64	(1,)	[42.86200733]
beta1	float64	(1,)	[-2.3328211]
beta2	float64	(1,)	[0.04354689]

In the following figure true labels, predicted labels from linear regression and predicted labels from polynomial regression are given in the same plot.



As seen in the figure poltnomial regression fits the data better because of the spahe of the true labels are more likely to be fitted by second degre curve than a linear line. The MSE values are also validates this truht. As seen in the following figure the MSE of polynomial regression model is less than the linear regression model as expected because of the reason shape of the original labels.

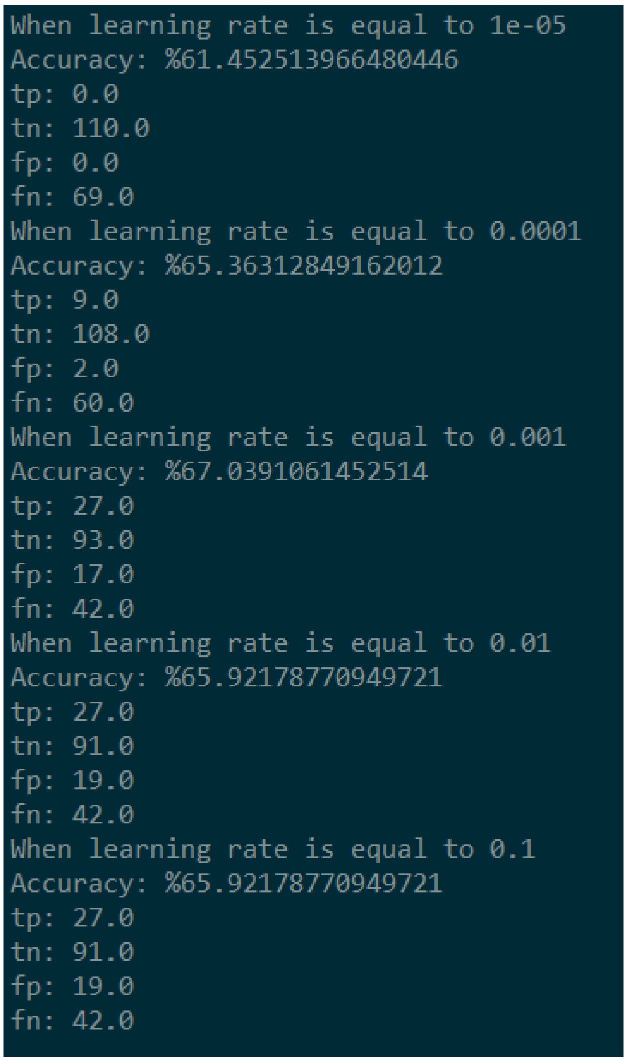


```
MSE of the linear model is 38.48296722989415
MSE of the polynomial model is 30.330520075853713
```

Question 3

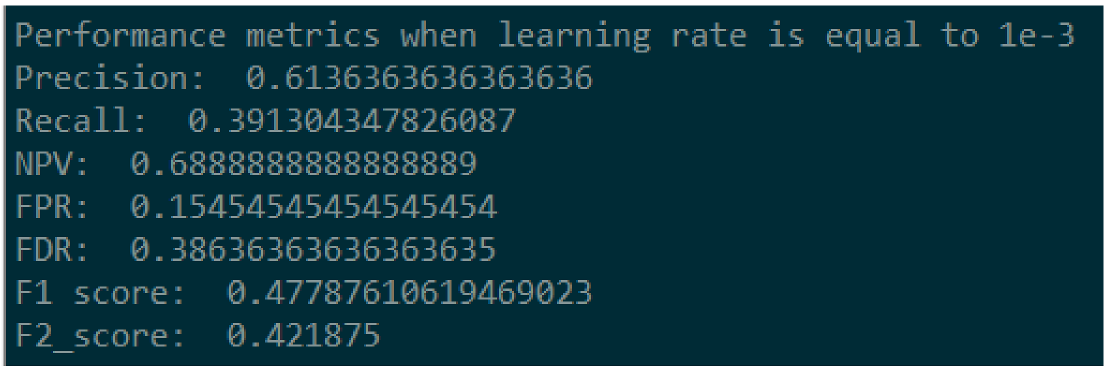
3.1

After implementing the full batch gradient ascent algorithm ı tried different learning rates which are $[10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}]$ and the accuricies are reported in the following figure.



```
When learning rate is equal to 1e-05
Accuracy: %61.452513966480446
tp: 0.0
tn: 110.0
fp: 0.0
fn: 69.0
When learning rate is equal to 0.0001
Accuracy: %65.36312849162012
tp: 9.0
tn: 108.0
fp: 2.0
fn: 60.0
When learning rate is equal to 0.001
Accuracy: %67.0391061452514
tp: 27.0
tn: 93.0
fp: 17.0
fn: 42.0
When learning rate is equal to 0.01
Accuracy: %65.92178770949721
tp: 27.0
tn: 91.0
fp: 19.0
fn: 42.0
When learning rate is equal to 0.1
Accuracy: %65.92178770949721
tp: 27.0
tn: 91.0
fp: 19.0
fn: 42.0
```

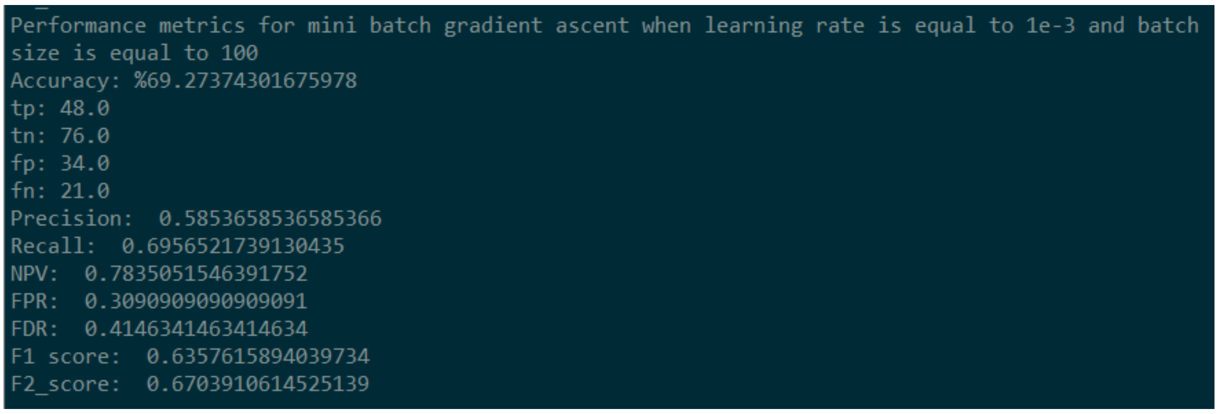
As seen in the figure the highest accuracy is reported when learning rate is equal to 0.001. The performance metrics when learning rate is equal to 0.001 is reported in the following figure.



```
Performance metrics when learning rate is equal to 1e-3
Precision:  0.6136363636363636
Recall:  0.391304347826087
NPV:  0.6888888888888889
FPR:  0.15454545454545454
FDR:  0.38636363636363635
F1 score:  0.47787610619469023
F2_score:  0.421875
```

3.2

After implementing mini-batch gradient ascent algorithm with batch size = 100 the following results are obtained.



```
Performance metrics for mini batch gradient ascent when learning rate is equal to 1e-3 and batch
size is equal to 100
Accuracy: %69.27374301675978
tp: 48.0
tn: 76.0
fp: 34.0
fn: 21.0
Precision:  0.5853658536585366
Recall:  0.6956521739130435
NPV:  0.7835051546391752
FPR:  0.3090909090909091
FDR:  0.4146341463414634
F1 score:  0.6357615894039734
F2_score:  0.6703910614525139
```

Since we are randomly splitting the train dataset in each iterations in mini batch gradient ascent accuricies can slightly be change in every run.

3.3

Precision, recall, and accuracy are not strong enough measures when the classes for classification tasks are unequally distributed, since if one class has a large numerical advantage over the other, that class's accuracy will dominate, while the other may be severely mispredicted. In a binary classification on the other hand, if a classifier predicts everything to be positive and positive data are dominating, recall and accuracy will be quite high, but the classifier will be ineffectual if it also identifies the negatives poorly. FPR and FDR, on the other hand, would assist in a better understanding of the performance.

Codes

Q1

```
#import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


#read data as matrix form from csv

images = pd.read_csv("images.csv")

images = images.values

meaned_images = images - np.mean(images,axis = 0)




def PCA (images, num_principle_components):

   pve = []

   #Mean center the data

   meaned_images = images - np.mean(images,axis = 0)

   #Compute covariance matrix

   covar_matrix = np.cov(meaned_images,rowvar  = False) #set rowvar to False to get covariance
matrix in required dims

   #Calculate eigenvalues and eigenvectors of covcariance matrix

   eigen_values, eigen_vector = np.linalg.eigh(covar_matrix)

   #sorting eigen vectors and eigen values

   sorted_index = np.argsort(eigen_values)[::-1]

   sorted_eigen_vectors = eigen_vector[:,sorted_index]
```

```
    sorted_eigen_values = eigen_values[sorted_index]

    #select first n principle components

    principle_components = sorted_eigen_vectors[:,0:num_principle_components]

    eigen_values = sorted_eigen_values[:num_principle_components]

    #calculate PVE

    for i in eigen_values:

        pve.append(i/sum(sorted_eigen_values)*100)

    return principle_components,pve


def reshape_components(principle_components,num_principle_components):

    reshaped_principle_components = []

    for i in range (num_principle_components):

        pca = np.reshape(principle_components[:,i],(48,48))

        reshaped_principle_components.append(pca)

    return reshaped_principle_components


def show_pca(num_principle_components):

    plt.figure(1)

    for i in range(num_principle_components):

        plt.subplot(2, 5, i+1)

        plt.imshow(reshaped_components[i],cmap="gray")


def print_pve(pve):

    j = 1

    for i in pve :
        print("PVE of principle component "+str(j)+" is % "+str(i))

        j+=1


def plot_pve():

    num_principle_components = [1, 10, 50, 100, 500]

    pves = []

    for k in num_principle_components:

        principle_components,pve = PCA (images, k)

        pves.append(sum(pve))

        print("for k equals "+str(k)+" pve is % "+ str(sum(pve)))

    plt.figure(2)

    plt.scatter(num_principle_components,pves,color = "blue")

    plt.title("k vs PVE")

    plt.xlabel("k")

    plt.ylabel("PVE")

    plt.show()


def reconstruct_image():

    num_principle_components = [1, 10, 50, 100, 500]

    i=1

    plt.figure(3)

    for k in num_principle_components:

        principle_components,pve = PCA (images,k)

        eigen_vec = principle_components[:,:k]

        eigen_k_transpose = eigen_vec.transpose()
        project = np.dot(meaned_images[0], eigen_vec)

        recon = np.dot(project,eigen_k_transpose)

        recon = np.add(recon, np.mean(meaned_images, axis = 0))

        plt.subplot(1, 5, i)

        plt.imshow(np.reshape(recon,(48,48)),cmap="gray")

        i+=1


principle_components,pve = PCA (images,10)

reshaped_components = reshape_components(principle_components,10)

print_pve(pve)

show_pca(10)

plot_pve()

reconstruct_image()
```

Q2

```
#import libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt


"""

Part 1 Linear Regression


"""
```

```
#read data as matrix form from csv

features_df = pd.read_csv("question-2-features.csv")

features = features_df.values

labels_df = pd.read_csv("question-2-labels.csv")

labels = labels_df.values


#calculate X^TX matrix

xtx = np.dot(features.transpose(),features)

rank = np.linalg.matrix_rank(xtx)


#exract LSTAT to use only feature for training

lstat = features_df[["LSTAT"]].values

train_features = np.ones((506,2))

train_features[:,1] = lstat[:,0]


#calculate B=(X^TX)^-1(X^TY)

reverse_xtx = np.linalg.inv(np.dot(train_features.transpose(),train_features))

xty = np.dot(train_features.transpose(),labels)

beta0,beta1 = np.dot(reverse_xtx,xty)


#predicted_labels = np.dot(lstat,beta)

predicted_labels_lin = lstat*beta1+beta0


#calculate the MSE

mse= np.sum((labels - predicted_labels_lin)**2) / len(labels)
```

```
print("MSE of the linear model is "+ str(mse))


"""

Part 2 Polynomial Regression


"""

#exract LSTAT to use only feature for training

train_features = np.ones((506,3))

train_features[:,1] = lstat[:,0]

train_features[:,2] = (lstat[:,0])**2

#calculate B=(X^TX)^-1(X^TY)

reverse_xtx = np.linalg.inv(np.dot(train_features.transpose(),train_features))

xty = np.dot(train_features.transpose(),labels)

beta0,beta1,beta2 = np.dot(reverse_xtx,xty)


#predicted_labels = np.dot(lstat,beta)

predicted_labels_pol = (lstat**2)*beta2+lstat*beta1+beta0


#calculate the MSE

mse= np.sum((labels - predicted_labels_pol)**2) / len(labels)

print("MSE of the polynomial model is "+ str(mse))


#Plot ground truth labels and predicted labels

plt.scatter(lstat,labels , color = "blue")

plt.scatter( lstat,predicted_labels_pol, s=50,color = "red")
plt.plot( lstat,predicted_labels_lin, color = "orange")

plt.title("Price vs LSTAT")

plt.xlabel("LSTAT")

plt.ylabel("Price")

plt.legend(["predicted labels(linear regression)",'true labels','predicted labels(polynomial
regression)'])

plt.show()
```

Q3

```
#import libraries

import pandas as pd

import numpy as np

import random


train_features = pd.read_csv("question-3-features-train.csv").values

test_features = pd.read_csv("question-3-features-test.csv").values

train_labels = pd.read_csv("question-3-labels-train.csv").values

test_labels = pd.read_csv("question-3-labels-test.csv").values


def find_min_max(dataset):

    minmax = list()

    for i in range(len(dataset[0])):

        col_values = []

        for row in dataset:
            col_values.append(row[i])
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax
#normalize dataset
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])

normalize_dataset(train_features,find_min_max(train_features))
normalize_dataset(test_features,find_min_max(test_features))

def sigmoid(z):
    sig = (1/(1+np.exp(-z)))
    return sig

def model_fit_full(features, labels, rate,iterations):
    weights = np.zeros(len(features[0])+1)
    beta0 = weights[0]
    betai = weights[1:len(weights)]
    for iteration in range(iterations):
        z = beta0 + np.dot(features, betai)
        predicted_labels = sigmoid(z)
        error = np.subtract(train_labels,predicted_labels)[:,0]

        dgradient = np.dot(features.T, error)

        beta0 += rate * np.sum(error)

        betai += rate * dgradient

    predicted_weights = np.append(beta0, betai)

    return predicted_weights


def create_subsets(features,labels,batch_size):

    #creates random subsets of train dataset for mini batch gradient descent where size of subset
equals to 100

    index = random.randint(0,612)

    mini_train_features = train_features[index:index+batch_size]

    mini_train_labels = train_labels[index:index+batch_size]

    return mini_train_features,mini_train_labels


def model_fit_mini(features, labels, rate,iterations,batch_size):

    weights = np.random.normal(0, 0.1, len(features[0])+1)

    beta0 = weights[0]

    betai = weights[1:len(weights)]

    for iteration in range(iterations):

        mini_train_features,mini_train_labels = create_subsets(features,labels,batch_size)

        z = beta0 + np.dot(mini_train_features, betai)

        predicted_labels = sigmoid(z)

        error = np.subtract(mini_train_labels,predicted_labels)[:,0]

        dgradient = np.dot(mini_train_features.T, error)
        beta0 += rate * np.sum(error)

        betai += rate * dgradient

    predicted_weights = np.append(beta0, betai)

    return predicted_weights


def predict(test_features,weights):

    beta0 = weights[0]

    betai = weights[1:len(weights)]

    predicted_labels = []

    for c in range(len(test_features)):

        score = beta0 + np.sum(np.dot(test_features[c], betai))

        prob_true = sigmoid(score)

        prob_false = 1-sigmoid(score)

        if prob_false >= prob_true:

            prediction = 0

        else:

            prediction = 1

        predicted_labels.append(prediction)

    return predicted_labels


def calculate_accuracy(predicted_labels,test_labels):

    tp = 0.0 #true positive count

    tn = 0.0 #true negative count

    fp = 0.0 #false positive count

    fn = 0.0 #false negative count
    for i in range(len(predicted_labels)):
        prediction = predicted_labels[i]
        actual = test_labels[i]
        if ((prediction == 1) & (actual == 1)):
            tp+=1
        elif ((prediction == 1) & (actual == 0)):
            fp+=1
        elif ((prediction == 0) & (actual == 0)):
            tn+=1
        elif ((prediction == 0) & (actual == 1)):
            fn+=1
    accuracy = ((tp+tn)/(tp+tn+fp+fn))*100
    return accuracy,tp,tn,fp,fn


def performance_metrics(tp,tn,fp,fn):
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    NPV = tn/(tn+fn)
    FPR = fp/(fp+tn)
    FDR = fp/(fp+tp)
    F1_score = 2*precision*recall/(precision+recall)
    F2_score = 5*precision*recall/(4*precision+recall)
    print("Precision: ",precision)
    print("Recall: ",recall)
    print("NPV: ",NPV)
    print("FPR: ",FPR)

    print("FDR: ",FDR)

    print("F1 score: ",F1_score)

    print("F2_score: ",F2_score)



learning_rates = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1]

for rate in learning_rates:

    weights = model_fit_full(train_features, train_labels,rate,1000)

    predicted_labels = predict(test_features,weights)

    accuracy,tp,tn,fp,fn = calculate_accuracy(predicted_labels,test_labels)

    print("When learning rate is equal to "+str(rate))

    print("Accuracy: %"+str(accuracy))

    print("tp: "+str(tp))

    print("tn: "+str(tn))

    print("fp: "+str(fp))

    print("fn: "+str(fn))


print("Performance metrics for full gradient ascent when learning rate is equal to 1e-3")

weights = model_fit_full(train_features, train_labels,1e-3,1000)

predicted_labels = predict(test_features,weights)

accuracy,tp,tn,fp,fn = calculate_accuracy(predicted_labels,test_labels)

performance_metrics(tp,tn,fp,fn)

print("Performance metrics for mini batch gradient ascent when learning rate is equal to 1e-3 and
batch size is equal to 100")
```

```
weights = model_fit_mini(train_features, train_labels,1e-3,1000,100)

predicted_labels = predict(test_features,weights)

accuracy,tp,tn,fp,fn = calculate_accuracy(predicted_labels,test_labels)

print("Accuracy: %"+str(accuracy))

print("tp: "+str(tp))

print("tn: "+str(tn))

print("fp: "+str(fp))

print("fn: "+str(fn))

performance_metrics(tp,tn,fp,fn)
```