Batu Arda Düzgün 21802633
EEE-482 HW-3

Question 1:

In this question, we are given Blood-oxygen level-dependent (BOLD) responses of a neural population in the human visual cortex as Yn that represents 1000 response samples, and Xn that represents 100 regressors that may explain the responses. For all parts, the proportion of explained variance (R^2) should be calculated as the square of Pearson's correlation coefficient between measured and predicted responses.

a) Use the ridge regression method to fit regularized linear models to predict noisy BOLD responses as a weighted sum of given regressors. Perform 10-fold cross-validation to tune the ridge parameter ($\lambda \in [0\ 10^{12}]$) based on model performance. For each cross-validation fold, do a three-way split of the data: select a validation set of 100 contiguous samples, a testing set of 100 samples (that immediately precede the validation set assuming circular symmetry), and a training set of length 800 samples. Fit a separate model for each λ using the training set. Find R^2 of each model on the testing set. Separately estimate R^2 of each model on the validation set. Plot the average R^2 across cross-validation folds, measured on the testing set as a function of λ. Find the optimal ridge parameter λ_{opt} that maximizes average R^2. Find the model performance by calculating the average R^2 across cross-validation folds, measured on the validation set for λ_{opt}. Plot R^2 curves obtained on testing and validation data for all λ values. Interpret the results.

First of all, in this problem, the problem doesn't use the common naming convention where the validation set is the set used to find the best hyperparameters, and the test set is the set used to find the performance of the model. It uses the opposite of it where the test set is used to find the hyperparameters and the validation set is used to find the model's performance. I wanted to use the common naming convention, so I used the validation set to calculate the hyperparameters, which is only λ in this problem, and the test set to find the model performance measurement.

The formula for the ridge regression is given in equation 1. Where y is the output response vector, X is the regressor matrix, and w is the weights vectors.

$$\min\left[(y - Xw)^2 + \lambda(w)^2\right] \tag{1}$$

The solution to equation w in equation 1 is given in equation 3.

$$(X^T X + \lambda I)w = X^T y \tag{2}$$

$$w = (X^T X + \lambda I)^{-1} X^T y \tag{3}$$

Ridge regression compared to linear regression also fits the model for minimum squared error but it also tries to minimize the square of the weight parameters as it can be seen in equation 1. This way ridge regression tries to prevents the model from overfitting in the given training set. To calculate ridge regression for all the training sets with the different λ values and for the rest of the computation in the homework I used Python 3.8, python works almost fully on library's and without them it almost doesn't have any functions for the scientific calculations we what we want to do here. Because of this I imported the related packages to do the tasks we are given. I paid attention to not use any library that doesn't already come with the Anaconda install. So, all the libraries in this code already should come with python and they shouldn't require an extra download. I wrote the import part of my code down below. If your python doesn't have the used packages and you have pip, you can install the missing packages with command such as "python -m pip install numpy, python -m pip install scipy or python -m pip install matplotlib."

```
import sys # the primer already has it

import numpy as np # this brings simple matrix operations to python
import matplotlib.pyplot as plt # this brings nice loking plots to python

from scipy.stats import norm # used to calculet CDF's used to calculate the p
#value
import scipy.io as sio # this is used to load MATLAB files
```

Now if we return to the question given in part a. First, I defined a ridge regression function. This function calculates the weight vector using ridge regression for a input matrix, output vector pair with a ridge parameter λ. I does this by simply doing the matrix operation given in equation 3. This function is called thousands of times during this question so it is better to write it as a separate function. The code for ridge regression function is given below.

```

def ridge_regression(X, y, lamda):
    return np.linalg.inv(X.T.dot(X) + lamda * np.identity(np.shape(X)[1])).dot(X.
T).dot(y) # matrix operation which gives the weight vector for the ridge regesion
.
```

The question also wants us to calculate R^2 as the square of Pearson's correlation coefficient between measured and predicted responses. The equation for 2 random variables A and B, is given below.

$$p = \frac{con(A,B)}{\sigma_A \sigma_B} \tag{4}$$

So, the proportion of explained variance (R^2) is calculated as shown below for the rest of the homework.

$$R^2 = p^2 = \left(\frac{con(A,B)}{\sigma_A \sigma_B}\right)^2 \tag{5}$$

To calculate equation 5, I used the np.corrcoef function, which generates the correlation matrix between the given parameters. I pick the correlation value between the two parameters and square its value to calculate R^2. This function is used for the rest of the homework for R^2 computations.

Then the question also wants us to do 10-fold cross-validation with a 3-way split which separates the given data to a validation set of 100 contiguous samples, a testing set of 100 samples that immediately precede the validation set assuming circular symmetry, and a training set of length 800 samples. This means we will have 10 cross-validation folds. For each of them, the training set will be used to fit a model for the given λ value. Then the validation set will be put into this model and the model's proportion of explained variance (R^2) will be calculated. Then the terms of the validation set will be concatenated to the beginning of training set. We add it to the beginning because, assuming circular symmetry, the validation set always immediately precedes the training set. Then this new set with 900 elements will be fit into a model using ridge regression for the same given λ value. Then the test set will be put into this model and the model's proportion of explained variance (R^2) will be calculated. Finally, the 10 R^2 values of the validation set will be averaged and the 10 R^2 values of the test set will be averaged. The code for this cross-validation function is given below.

```
def cross_validationation(X, y, K, lamda):

    fold_size = int(np.size(y)/K) # size of test and validationation


    validation_R2 = 0 # difrent R^2 will be summed up in here and then will be
#devided by K to find the average
    test_R2 = 0

    for i in range(K):

        train_data_ind, test_data_ind, validation_data_ind = [], [], []

        test_data_start = i * fold_size # stating point of validation set for the
# given K
        validation_data_start = (i+1) * fold_size # stating point of test set for
# the given K
        train_data_start = (i+2) * fold_size # stating point of training set for
# the given K

        for j in range(2 * np.size(y)): # deciding which set every index in (0 ,
#999) will go to.

            if j in range(test_data_start, validation_data_start):
                test_data_ind.append(j % np.size(y))

            if j in range(validation_data_start, train_data_start):

                validation_data_ind.append(j % np.size(y))

            if  j in range(train_data_start, test_data_start + np.size(y)):

                train_data_ind.append(j % np.size(y))


        x_validation, x_test, x_training = X[validation_data_ind],
X[test_data_ind], X[train_data_ind] # palcing every input output pare in their
#respective set

        y_validation, y_test, y_training = y[validation_data_ind],
y[test_data_ind], y[train_data_ind]


        validation_weight = ridge_regression(x_training , y_training , lamda)
# riged rigresion to find the weight of the input parameters

        validation_R2 += (np.corrcoef(y_validation.transpose(), (x_validation.dot
(validation_weight)).transpose() )[0, 1]) ** 2 # corrcoef returns the coralation
#coaficent matrix of its inputs so here its outpıts [0 1] is the coralation
#coaficent coaficent betwen theh real y and the y we calculated form the fit
#model. we takes its square to calculate the R^2 of the model


        test_weight = ridge_regression(np.concatenate((x_validation, x_training)
, axis=0) , np.concatenate((y_validation, y_training) , axis=0) , lamda) # riged
#rigresion to find the weight of the input parameters for both the traing and the
# validation

        test_R2 += (np.corrcoef( y_test.transpose() , (x_test.dot(test_weight)).
transpose() )[0, 1]) ** 2 # again finding R^2 for the test.


    validation_R2 /= K # taking the R^2 average
    test_R2 /= K

    return validation_R2, test_R2
```

Again this function will be called thousands of times. Because of this, I wrote it as a separate function.

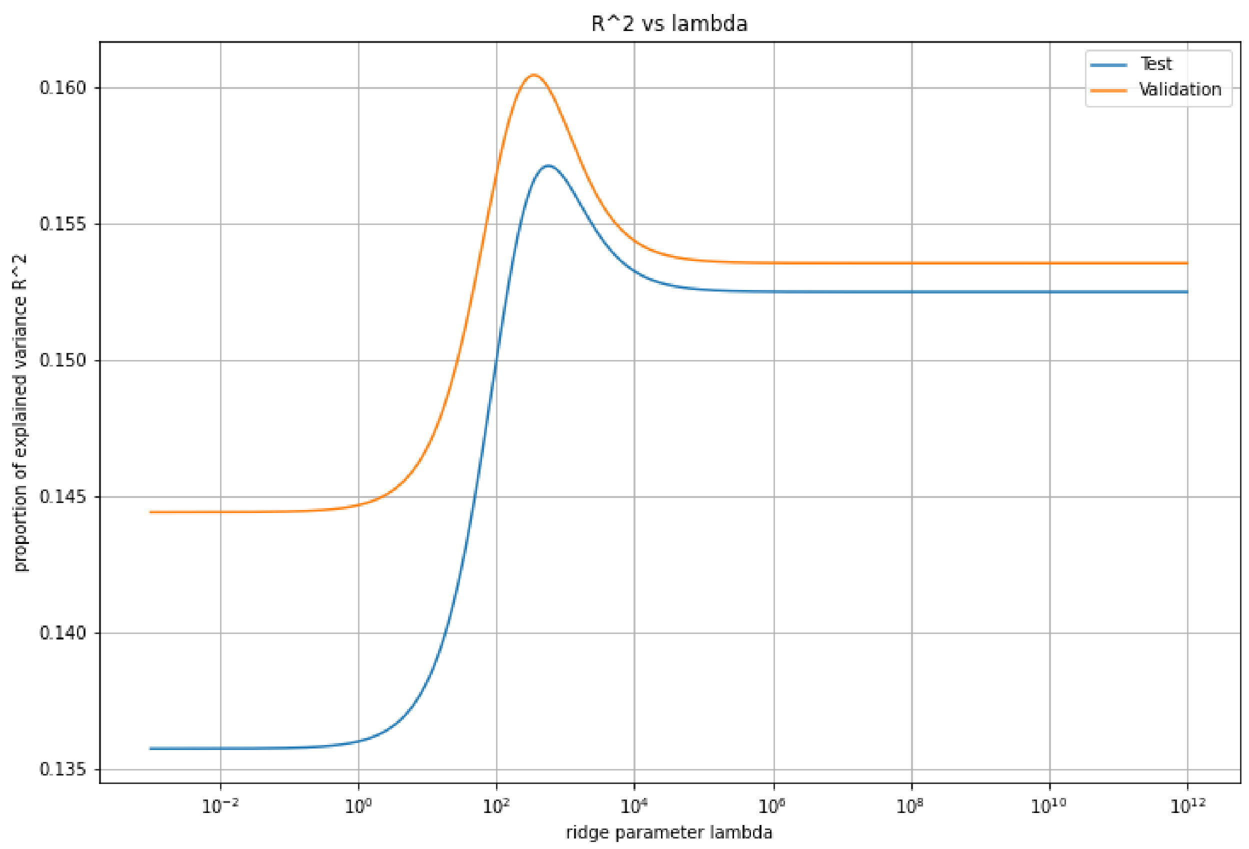
Now we have all the functions required for this part, we can begin to solve the question. The data for this homework is given to as MATLAB files. The files, have a too new format for the Pythons functions, which come with a standard Python install. To fix this, I opened the files on my MATLAB and re-saved them as a file format compatible with Python. Then I imported them into Python using the loadmat function. I used the same method and code for all parts of the homework to import the relevant data. The code for this is given below.

```
    data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
    Xn = data["Xn"]
    Yn = data["Yn"]
```

Now we can start solving the question. We are asked to test for ($\lambda \in [0\ 10^{12}]$) and vary the parameter logarithmically. For a logarithmically varying λ 0 is like negative infinity so I picked my λ for ($\lambda \in [10^{-3}\ 10^{12}]$). This doesn't matter that much because after 10^{-1} the R^2 value stops changing as we will see in the end of this part. I picked 1250 λ's which logarithmically vary for ($\lambda \in [10^{-3}\ 10^{12}]$) . Then for each λ I used the cross-validation function and added the R^2 values of validation and test into separate lists. Then I plotted the R^2 values obtained on testing and validation data versus my λ values. Then I picked the λ value, which gives the highest R^2 value for the validation data as the λ_{opt}. I didn't pick it from the test data's R^2 curve because If we did, we wouldn't be able to find a model performance measurement because there will not be any data left to calculate it. So the parameters should be chosen for the output of the validation data. Then finally I looked at the test data's R^2 curves value at λ_{opt} to find the model performance average R^2. The code which does this is given below.

```
    lamda_values = np.logspace(-
3, 12, num=1250, base=10) # difrent lambda values to calculate proportion of 
#explained variance

    validation_R2_for_lamda_values = [] # we will fill these with the R^2 values 
#for all the lambda
    test_R2_for_lamda_values = []


    for lamda in lamda_values: # doing the K fold regresion for each of the 
#lambda values

        validation_R2 , test_R2 = cross_validationation(Xn, Yn, 10, lamda)

        validation_R2_for_lamda_values.append(validation_R2)
        test_R2_for_lamda_values.append(test_R2)


    plt.figure(figsize=(12, 8)) # plotting R^2 curves
    plt.plot(lamda_values, test_R2_for_lamda_values)
    plt.plot(lamda_values, validation_R2_for_lamda_values)
    plt.legend(['Test', 'Validation',])
    plt.ylabel('proportion of explained variance R^2')
    plt.xlabel('ridge parameter lambda')
    plt.title('R^2 vs lambda')
    plt.xscale('log')
    plt.grid()


    max_R2 =  max(validation_R2_for_lamda_values)

    lamda_opt_index = validation_R2_for_lamda_values.index(max_R2)

    performance_of_lambda_opt = test_R2_for_lamda_values[lamda_opt_index]
# finding the model performance for the optimum lambda

    print('optimal ridge parameter across cross-
validation folds, measured on the validation set =' + str(lamda_opt) + " and the
model with this lambda, gives a model performance of R^2 = " +
str(performance_of_lambda_opt))
```

This gives this graph as its output.



We can see the general shape of the two R^2 curves look very much each other. And the λ_{opt} which gives the peak value of the validation R^2 curve also gives a high R^2 for the test R^2 curve. Like I said before, we shouldn't pick the peak of the test R^2 curve. Also, something which is interesting is the R^2 values for the validation R^2 curve has higher R^2 values than the test R^2 curve. This is unexpected because the models used to create the test R^2 curve is better as it is created from 900 sample points instead of the 800 sample points used to create the models used in the validation models. Also, it finds the λ_{opt} as 353.5402, which gives a model performance of $R^2 = 0.1566$.

$$\lambda_{opt} = 353.54$$

$$model\ performance\ for\ \lambda_{opt}: R^2 = 0.1566$$

b) Determine confidence intervals for parameters of the OLS model from part a (the model obtained for λ = 0). Generate bootstrap samples from the 1000 samples in the original data. Perform 500 bootstrap iterations, and refit a separate model at each iteration. Plot the mean and 95% confidence intervals of the parameters in the same graph. Identify and label on your plots, the model regressors which have weights that are significantly different than 0, at a significance level of $p < 0.05$.

Bootstrap is a method to create new sample sets from an already existing sample set. For bootstrap of n iterations on a sample set with m elements, we pick m elements from the sample set with replacement and create a new bootstrap set. We repeat this process to n times to create n different (or not different depending on luck) sample sets. Then we can use these new sample sets for whatever we want. Doing bootstrap of n iteration can be seen as doing the same experiment n times so it shows the inherent randomness of the experiment and ir is a good tool to test the reliability of a model.

It is important to note that bootstrap is a process that uses random sampling, so any time we use it, we will get a slightly different plot but it will still have a consistent general shape mean and variance. I did not want to use any seed in my code for the random parts, so the code's output will be slightly different every time it is run for question 1 part b, c and all of question 2.

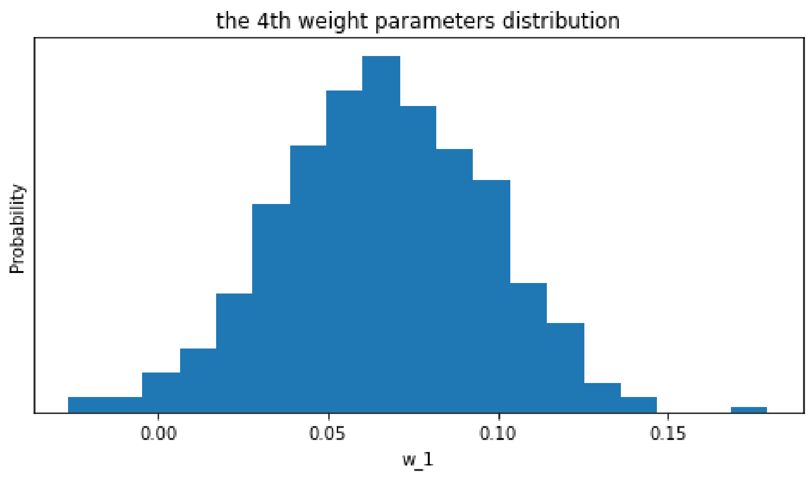
For this part, I created a bootstrap function that generates n bootstraps for a given sample set and λ value. Using the ridge regression function defined in part a it calculates the weight parameters of all n of its bootstrap iterations and returns the weights of all of the iterations as a matrix. The code for this function is given below.

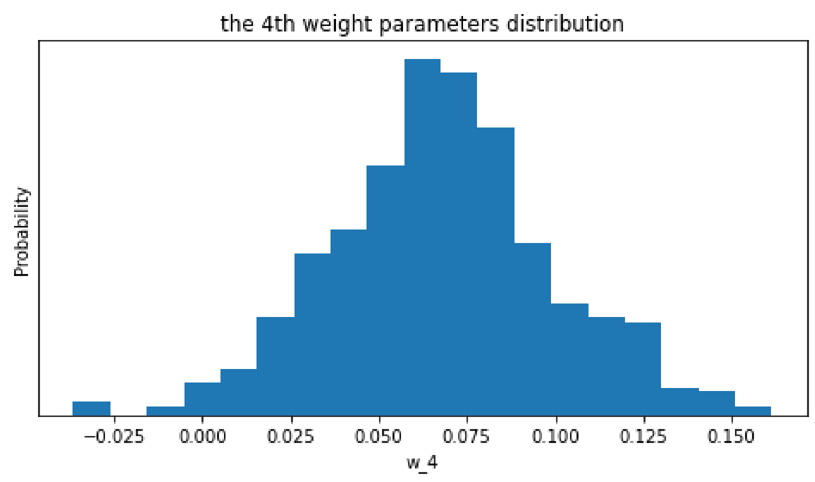
```
def bootstrap(iteration_count, X, y, lamda): # the function to generate input
#weights for n bootstraps for a given input, output and ridged regreasion

    weights = [] # we will fill this in

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.size(y), np.size(y)) # chosing the
# index of the input output pairs which will be added to this iteration

        X_iteration = X[bootstrap_indexs]  # adding the choosen input output
#pairs in to the iteration
        y_iteration = y[bootstrap_indexs]

        weights.append(ridge_regression(X_iteration, y_iteration, lamda))
# adding the weights for this iteration to the output list

    return weights
```

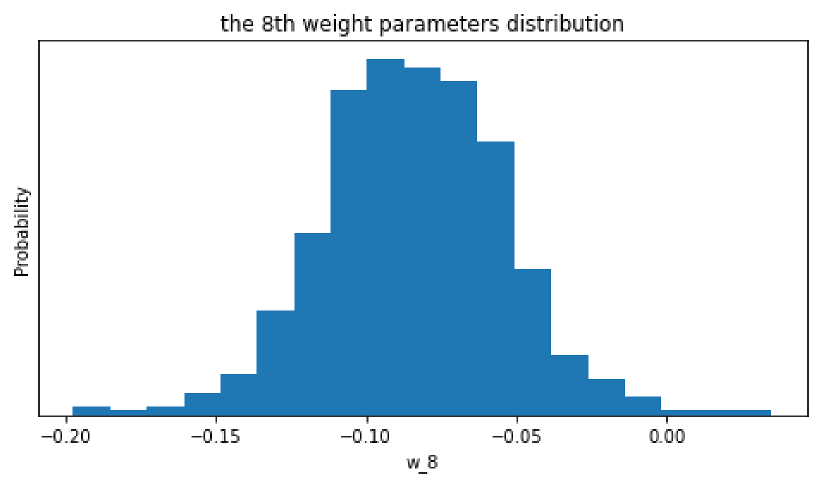
Again I load the data given to us in to Python then put the Xn and Yn in to bootstrap function with $\lambda = 0$ for 500 iterations. Then I plotted the distribution of some of the weights to show they look like a gaussian. The code for this is given below.

```
    data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
    Xn = data["Xn"]
    Yn = data["Yn"]
    weights = np.array(bootstrap(500, Xn, Yn, 0)) # getting the weights form 500 
bootstrap iterations

    plt.figure() # we know this distribution should look like a gausian but I wan
ted to show it looks like a gausian too.
    plt.title('the first weight parameters distribution for lambda = 0')
    plt.xlabel('w_1')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,0], bins=19, density=True)

    plt.figure() # we know this distribution should look like a gausian but I wan
ted to show it looks like a gausian too.
    plt.title('the 4th weight parameters distribution for lambda = 0')
    plt.xlabel('w_1')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,3], bins=19, density=True)

    plt.figure() # we know this distribution should look like a gausian but I wan
ted to show it looks like a gausian too.
    plt.title('the 8th weight parameters distribution for lambda = 0')
    plt.xlabel('w_1')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,7], bins=19, density=True)
```

this gives these output plots.







They look like Gaussians.

Then I computed the mean and standard deviation of each of the 100 weights. Then found the weights which are significantly different than 0, by calculating the two-tailed p-value of getting a zero from each of the weights distributions and finding the weights which have a p-value smaller the 0.05.

$$W_i \text{ is the random variable for the ith weight}$$

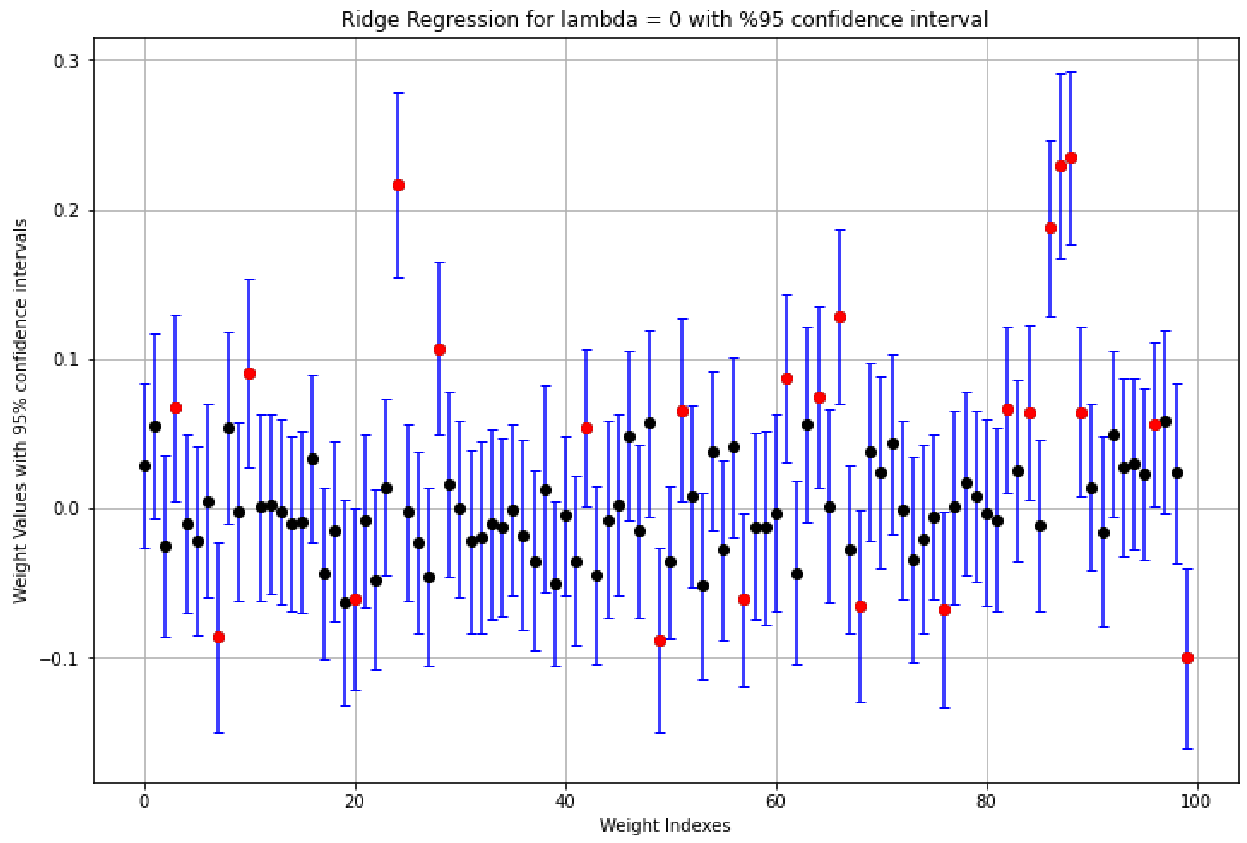
$$for\ E[W_i] > 0\,, p = 2 \cdot P(W_i < 0)$$

$$for\ E[W_i] < 0\,, p = 2 \cdot P(W_i > 0)$$

I plotted the weights and their 95% confidence intervals on the plot. Then I marked the weights which are significantly different from 0 with red. The code for this is given below.

```
    weights_mean = np.mean(weights, axis=0) # calculating the mean and standart
#deviation of each of the 100 weights
    weights_std = np.std(weights, axis=0)
    _95_percent_confidence = 2 * np.concatenate((weights_std.transpose(),
weights_std.transpose()) , axis = 0) # the weights are normal distribution so 2
#standart deviations to the both sides equals 95% confidence interval.

    p_values = 2 * (1 - norm.cdf(np.abs(weights_mean/weights_std))) # 2 tailed p
#value for the weight to have 0 mean

    significant_weights = np.where(p_values < 0.05) # finding the significant p
#values

    significant_weights = significant_weights[0]

    significant_weights_mean = weights_mean[significant_weights] # will be used
#in the plot


    plt.figure(figsize=(12, 8)) # plot of the weights and their 95% confidence
#intervals
    plt.grid() # grids so we can see the significant points are points which do
#not have a confidance interval which intercept 0
    plt.errorbar(np.arange(0,100), weights_mean , yerr = _95_percent_confidence ,
 ecolor='b', fmt='ok', capsize=3) # plotting all the weights
    plt.errorbar(significant_weights, significant_weights_mean, fmt='or')
# marking the significant weights with red.
    plt.ylabel('Weight Values with 95% confidence intervals')
    plt.xlabel('Weight Indexes')
    plt.title('Ridge Regression for lambda = 0 with %95 confidence interval')
```

This gives these output plots.



We can see from this, weights points which are significant are weights whose 95% confidence intervals do not cut the 0 line.

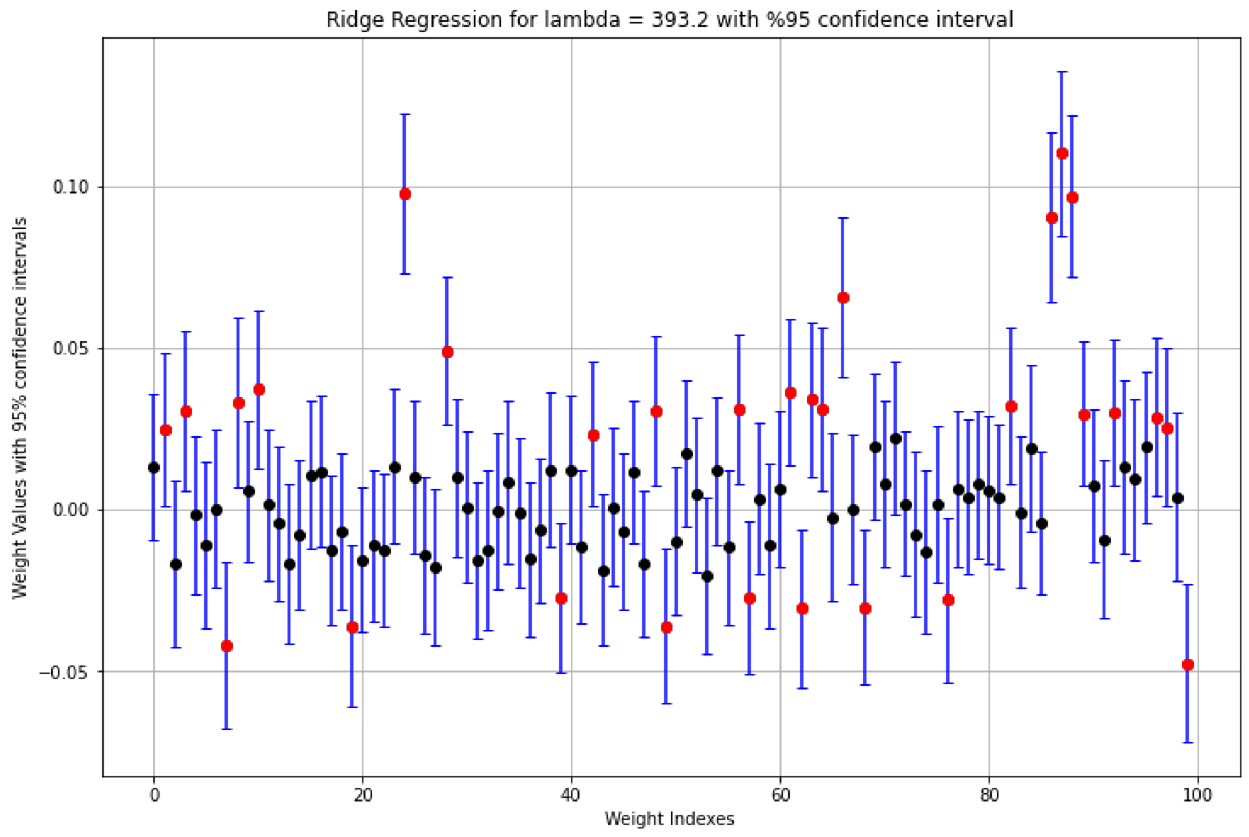
c) Determine confidence intervals for parameters of the regularized linear model from part a (the model obtained for λ = λ_{opt}). Generate bootstrap samples from the 1000 samples in the original data. Perform 500 bootstrap iterations, and refit a separate model at each iteration. Plot the mean and 95% confidence intervals of the parameters in the same graph. Identify and label on your plots, the model regressors which have weights that are significantly different than 0, at a significance level of $p < 0.05$. compare the result to part b.

I re-used the bootstrap function from part b in this part. I did the same operations as in the previous part. I first created the weights matrix for the 500 bootstraps but with λ = 353.5 this time. Then I found the significant weights the same way and plotted the weights with their 95% confidence interval, and marked the significant weights with red. The code for this is given below.

```
data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
Xn = data["Xn"]
Yn = data["Yn"]
```

```
    opt_lamda = 353.5 # I wrote this value by hand here because its computation
#takes a very long time and re calculating it here would be excessive

    weights = np.array(bootstrap(500, Xn, Yn, opt_lamda)) # getting the weights
#form 500 bootstrap iterations

    weights_mean = np.mean(weights, axis=0) # calculating the mean and standart
#deviation of each of the 100 weights
    weights_std = np.std(weights, axis=0)


    _95_percent_confidence = 2 * np.concatenate((weights_std.transpose(),
weights_std.transpose()) , axis = 0) # the weights are normal distribution so 2
#standart deviations to the both sides equals 95% confidence interval.


    p_values = 2 * (1 - norm.cdf(np.abs(weights_mean/weights_std))) # 2 tailed p
#value for the weight to have 0 mean


    significant_weights = np.where(p_values < 0.05) # finding the significant p
#values

    significant_weights = significant_weights[0]

    significant_weights_mean = weights_mean[significant_weights] # will be used
#in the plot



    plt.figure(figsize=(12, 8)) # plot of the weights and their 95% confidence
#intervals
    plt.grid() # grids so we can see the significant points are points which do
#not have a confidance interval which intercept 0
    plt.errorbar(np.arange(0,100), weights_mean , yerr = _95_percent_confidence ,
 ecolor='b', fmt='ok', capsize=3) # plotting all the weights
    plt.errorbar(significant_weights, significant_weights_mean, fmt='or')
# marking the significant weights with red.
    plt.ylabel('Weight Values with 95% confidence intervals')
    plt.xlabel('Weight Indexes')
    plt.title('Ridge Regression for lambda = ' + str(opt_lamda) +
' with %95confidence interval')
```

This prints the graph given below.



Again, we can see from this; significant weight points are weights whose 95% confidence intervals do not cut the 0 line. Ridge regression tries to minimize the squares of the weight parameters, we would expect this to decrease the number of weights which are non-zero. Because it decreases the means but this added constraint also decreases the variance of the weights. Thus, more weights who were close to becoming a significant weight became a significant weight in the ridge regression, and the significant weight number increased.

Question 2:

a) Responses from two separate populations of neurons are stored in the variables pop1 and pop2. We would like to examine whether the mean responses of the two populations are significantly different. The first population contains 7 neurons, whereas the second population contains 5 neurons. Using the bootstrap technique (10000 iterations), find the two-tailed p-value for the null hypothesis that the two datasets follow the same distribution.

Null hypothesis is, the two datasets follow the same distribution. Then for the distribution of the mean difference of pop1 and pop2 we can calculate the two-tailed p-value of getting the mean distribution of the original pop1, pop2. To do this I computed 10000 bootstraps like the question says and counted the number of them which fall in the rejection region of the distribution. The rejection region is mean differences which are absolutely greater then the given mean difference for the two-tailed p-value. Then to find the p-value I divided the count by the bootstrap count. I also fitted a gaussian to the distribution of the null-hypothesis and calculated the two-tailed p-value using CDF. This is explained more clearly as probabilistic equations down below.

$$A = mean\ difference\ distribution\ of\ pop1, pop2\ for\ the\ null\ hypothesis$$

$$k = E[pop1] - E[pop2]$$

$$p = 2 * P(A > |k|)$$

$$p = 2 * P\left(\frac{A - E[A]}{\sqrt{var(A)}} > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 2 * P\left(Z > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 2 * \left(1 - P\left(Z < \frac{|k| - E[A]}{\sqrt{var(A)}}\right)\right) \tag{6}$$

We first need to find the distribution of the difference pop1's and pop2's mean given they come from the same distribution. If they come from the same distribution, there is a pop3 composed of pop1 and pop2's elements, and we arbitrarily separated 7 of them as pop1 and the remaining 5 as pop2.

But first, we need a function to calculate the desired number of iterations of bootstrap for an input. For this, I created the bootstrap_singel function. This function returns the bootstraps as a matrix, and it will also be used in part c and e of this question. The code for the function is given below.

```
def bootstrap_singel(iteration_count, x): # the function to callculate n bootstra
ps of a vector

    x_bootstraps = []

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x))  #
 chosing the index of the values which will be added to this iteration

        x_bootstrap = x[bootstrap_indexs] # adding the choosen values in to the i
teration
```

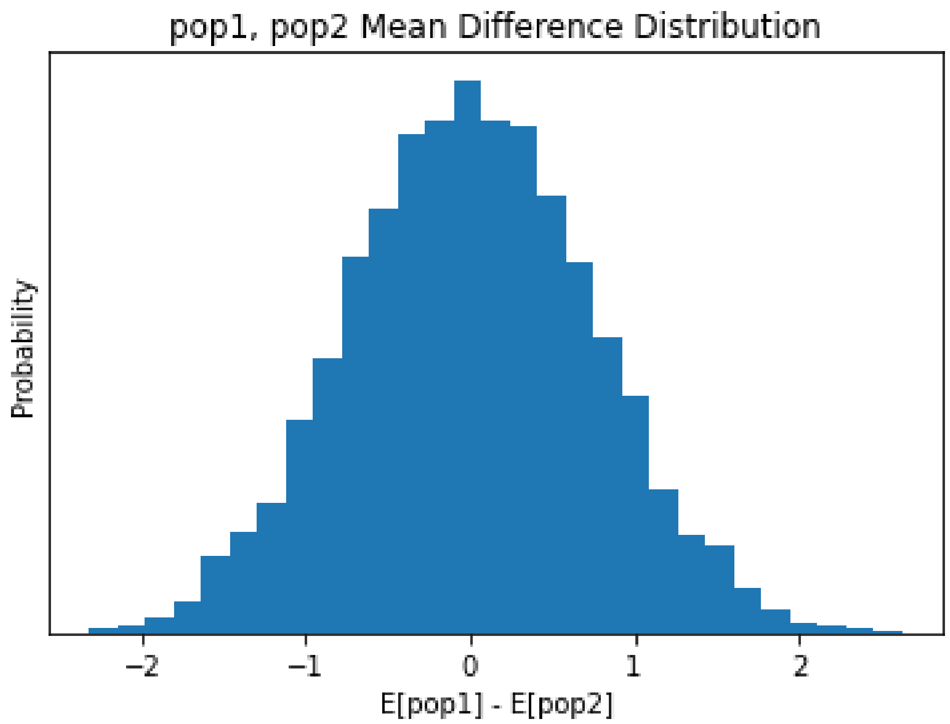
```
        x_bootstraps.append(x_bootstrap) # adding the bootstrap iteration to the 
output list

    return np.array(x_bootstraps)
```

Then, to find the null hypothesis's distribution, I concatenated pop1 and pop2 and used it as the input of the bootstrap_singel with an iteration count of 10000. Then I separated the first 7 elements of all 10000 iterations as a pop1 matrix and the last 5 elements of all 10000 iterations as a pop2 matrix. Then I found the pop1 and pop2 mean for all 10000 iterations. Then subtracted pop2 means from pop1 means to calculate the mean difference of all the iterations. These mean differences give the distribution of the null hypothesis. I plotted null hypothesis distribution to show how it looks like. I computed its sample mean and standard deviation. Then finally I counted the number of mean differences that fall in the rejection region and divided that number by 10000 to calculate the two-tailed p-value. Also using equation 6 I computed the two-tailed p-value. The code for this is given below.

```
def question2_part_a():


    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    pop1 = data2["pop1"]
    pop2 = data2["pop2"]

    # null hypothesis is they are from the same distribution. so we will add the 
#2 sets and create bootstraps from that and then we will take the first 7 as the 
#values in vox1 and the last 5 values as vox2. then we will compare their 
#difrence in means and look at how probable it is to get the mean difrence we get 
# in this particular case.

    pop1_pop2 = np.concatenate((pop1, pop2), axis = 1) # the neron group from our 
# null hypotezis

    bootstraps = bootstrap_singel(10000, pop1_pop2.transpose()) # creating 10000 
# bootstraps from the total population

    bootstrap_pop1 = bootstraps[:,0:7] # picking the first 7 elements as the pop1 
# for each of the 10000 bootstraps
    bootstrap_pop2 = bootstraps[:,7:12] # picking the last 5 elements as the pop2 
# for each of the 10000 bootstraps

    bootstrap_pop1_mean = np.mean(bootstrap_pop1, axis=1) # calculating mean of 
#pop1 for each of the 10000 bootstraps
```

```
    bootstrap_pop2_mean = np.mean(bootstrap_pop2, axis=1) # calculating mean of
#pop2 for each of the 10000 bootstraps

    bootstrap_mean_difrence = bootstrap_pop1_mean - bootstrap_pop2_mean
# calculating mean difrence for each of the 10000 bootstraps, this form the mean
#difference distribution of the population.

    plt.figure() # I wanted to show it looks like a gausian
    plt.title('pop1, pop2 Mean Difference Distribution')
    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difrence, bins=29, density=True)

    # now we will compute the 2 tailed p value of getting the mean difrence for
#the origanal pop1, pop2 given to us.

    given_mean_difrence = np.mean(pop1)-
np.mean(pop2) # computing the mean difrence for the origanal pop1, pop2 given to
#us

    p = 2 * (1 - norm.cdf(np.abs( (given_mean_difrence - np.mean(bootstrap_mean_d
ifrence)) / (np.std(bootstrap_mean_difrence)) ))) # here we compute P(|bootstrap_
#mean_difrence| > |given_mean_difrence|) by using the CDF of the standart normla
3distribution.

    significant_points = np.where(np.abs(bootstrap_mean_difrence) > np.abs(
given_mean_difrence)) # finding the significant points

    two_tailed_p_value = np.size(significant_points[0])/10000 # calcaulatin 2
#tailed p value

    print("Two-tailed p-
value for the null hypothesis that the two datasets follow the same distribution
is " + str(two_tailed_p_value) + " for counting bootstrap results over the given
difference.\nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and t
he 2 populations follow different distributions.")
```

this code gives the output plot given below.



We can see null-hypothesis distribution look like a gaussian. Also, this code finds the two-tailed p-value as 0.0085 by counting the number of points in the rejection region. This is a lot smaller than 0.05, so the null-hypothesis is wrong and the 2 populations are from different distributions. Also it finds the p-value as 0.0113 for fitting a gaussian to the distribution; these 2 p-values are a bit different but this is something expected because pop1 and pop2 are composed of 7 and 5 elements. Which means the number of elements is not sufficient to get the mean differences plot as a proper Gaussian.

b) BOLD responses recorded in two voxels in the human brain are stored in the variables vox1 and vox2. We would like to examine whether the voxel responses are similar to each other, by calculating their correlation. Using the bootstrap technique (10000 iterations), find the mean and 95% confidence interval of the correlation. Find the percentile of the bootstrap distribution, corresponding to a correlation value of 0.

Here I need to sample vox1 and vox2 identically to preserve the correlation between them. If I were to run the bootstrap function on them separately, they would show 0 expected correlation regardless of their correlation. So I defined a new bootstrap function to sample vox1 and vox2 identically. This function picks the same sample for vox1 and vox2 for each of the iterations. The code for this function is given below.

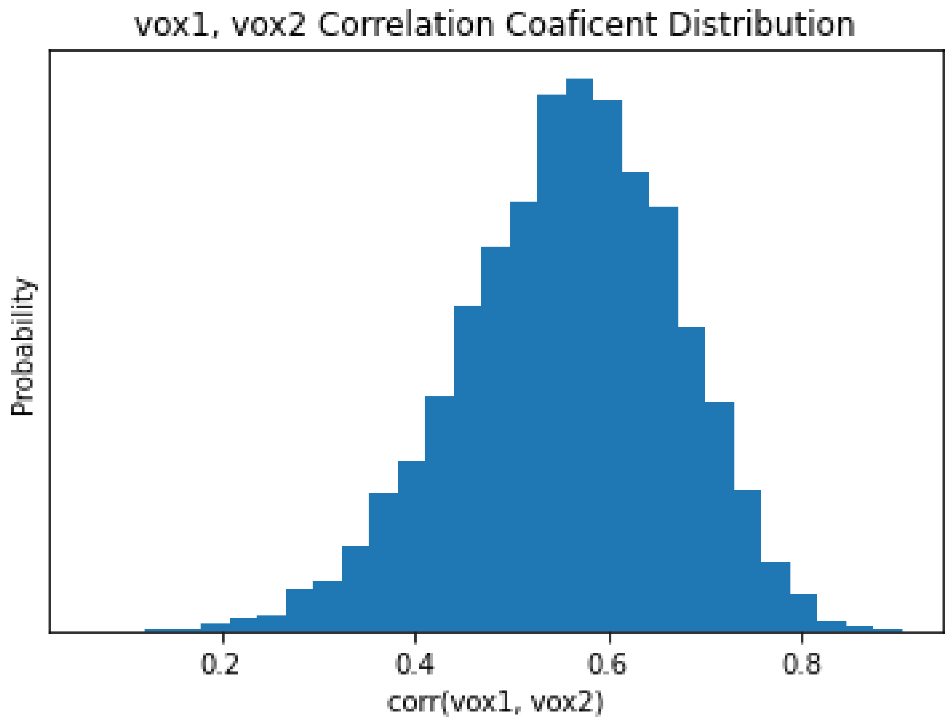
```
def bootstrap_double(iteration_count, x, y): # the function to callculate n
#bootstraps for 2 vectors with out breaking the coralation between them.
```

```
    x_bootstraps = []
    y_bootstraps = []

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x))
# chosing the index of the values which will be added to this iteration

        x_bootstrap = x[bootstrap_indexs] # adding the choosen values in to the
#iteration for x
        x_bootstraps.append(x_bootstrap) # adding the bootstrap iteration to the
#output list for x

        y_bootstrap = y[bootstrap_indexs] # adding the choosen values in to the
#iteration for y
        y_bootstraps.append(y_bootstrap) # adding the bootstrap iteration to the
#output list for y

    return np.array(x_bootstraps), np.array(y_bootstraps)
```

Then using this function, I calculated 10000 bootstraps of vox1, vox2 pairs. Then I calculated the correlation between each pair and acquired 10000 correlation values for the 10000 bootstraps. I drew a histogram of the correlation values to show their distribution. Then I calculated the sample mean of the correlation values by using the mean function. I couldn't use the 2 standard deviation equals 95% confidence interval trick because the distribution is not exactly gaussian. Because of this I found the 95% interval by ordering the correlation values and picking the 250th and 9750th ones, which are on the 2.5% and 97.5% borders. Finally, I found the percentile of correlation values corresponding to 0 by searching for correlation values absolutely smaller then 0.05. I didn't look for correlation values exactly equal to 0 because if I did I wouldn't be able to find any because the coralaiton values are floating point values. The code which does this is given below.

```
    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    vox1 = data2["vox1"]
    vox2 = data2["vox2"]

    bootstraps_vox1, bootstraps_vox2 = bootstrap_double(10000, vox1.transpose(),
vox2.transpose()) # creating 10000  bootstraps for vox1, vox2 pairs. they are row
# vectors and most fınctions are defiened for collum vector so we take their
#transpose.

    bootstraps_vox1_vox2_correlation = np.zeros(10000) # we will fill this in
    for i in range(10000): # Computing the correlation coaficent for 10000
#bootstrap vox1, vox2 pairs.

        bootstraps_vox1_vox2_correlation[i] = (np.corrcoef((bootstraps_vox1[i,:])
.transpose(), (bootstraps_vox2[i,:]).transpose() ))[0, 1] # comuting the
#correlation matrix for the ith bootstraps vox1 vox2 pair.

    plt.figure() # I wanted to show how the distribution looked
    plt.title('vox1, vox2 Correlation Coaficent Distribution')
    plt.xlabel('corr(vox1, vox2)')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstraps_vox1_vox2_correlation, bins=29, density=True)

    bootstraps_vox1_vox2_correlation_mean = np.mean(
bootstraps_vox1_vox2_correlation) # mean of the correlation

    # can use std to calculate because the distribution is not gausian.

    bootstraps_vox1_vox2_correlation_sorted = np.sort(
bootstraps_vox1_vox2_correlation) # we order the correlation the pick the 250th
#and 9750th term because they should be on the 2 borders of 95% interval.

    lower_95_percent_confidence_border = bootstraps_vox1_vox2_correlation_sorted[
249]
    upper_95_percent_confidence_border = bootstraps_vox1_vox2_correlation_sorted[
9749]

    zero_correlation_points = np.where(bootstraps_vox1_vox2_correlation_sorted <
0.05) # the correlation values are floting point so if we where to look at
#exeactly 0 we woulden't get any matches

    percenteg_of_zero_points = np.size(zero_correlation_points) * 100 / 10000
# turning the count to a percentage.

    print("Mean of the correlation coefficient distribution = " + str(
bootstraps_vox1_vox2_correlation_mean))
    print("95% confidence interval of the correlation coefficient distribution =
(" + str(lower_95_percent_confidence_border) + ", " + str(
upper_95_percent_confidence_border) + ")")
    print("Percentile of bootstrap distribution, corresponding to a correlation
value of 0 = " + str(percenteg_of_zero_points) + "%")
```

This code gives the output plot of:



It also finds the mean of the correlation values as 0.5589 and the 95% confidence interval as (0.323, 0.757). Also, the bootstrap distribution percentile, corresponding to a correlation value of 0 is 0%.

$$Mean = 0.5589$$

$$95\%\ confidance\ interval = (0.323, 0.757)$$

$$Percentile\ of\ the\ distribution, corresponding\ to\ a\ correlation\ value\ of\ 0\ =\ 0.0\%$$

c) Note that estimation of confidence intervals and hypothesis testing are dual problems. For the dataset examined in part b, use bootstrapping (10000 iterations) to simulate the distribution of the null hypothesis that two voxel responses have zero correlation. Find the one-tailed p-value for the two voxel responses having zero or negative correlation. Compare this to the result in part b.

The null hypothesis is vox1 and vox2 have 0 correlation. To get the distribution of this null hypothesis, I will sample vox1 and vox2 separately for the bootstraps. This way, the correlation between vox1 and vox2 will be broken and will give us the desired distribution. To create these bootstraps, I re-used the function defined in part a and created 10000 bootstraps of vox1 and 10000 bootstraps for vox2 separately. Then I calculated the correlation value between each vox1, vox2 pair and acquired 10000 correlation values. These values are the distribution of the null hypothesis. I plotted this distribution to show it. Then I computed the one-tailed p-value for

2 voxels having 0 or negative correlation by counting the number of bootstraps which fall in the rejection region and dividing that number with 10000. I also computed the one-tailed p-value by fitting a Gaussian to the distribution of the null-hypothesis and using equation 7.

$$A = mean\ difference\ distribution\ of\ vox1, vox2\ for\ the\ null\ hypothesis$$

$$k = corr(vox1, vox2)$$

$$p = P(A > k)$$

$$p = P\left(\frac{A - E[A]}{\sqrt{var(A)}} > \frac{k - E[A]}{\sqrt{var(A)}}\right)$$

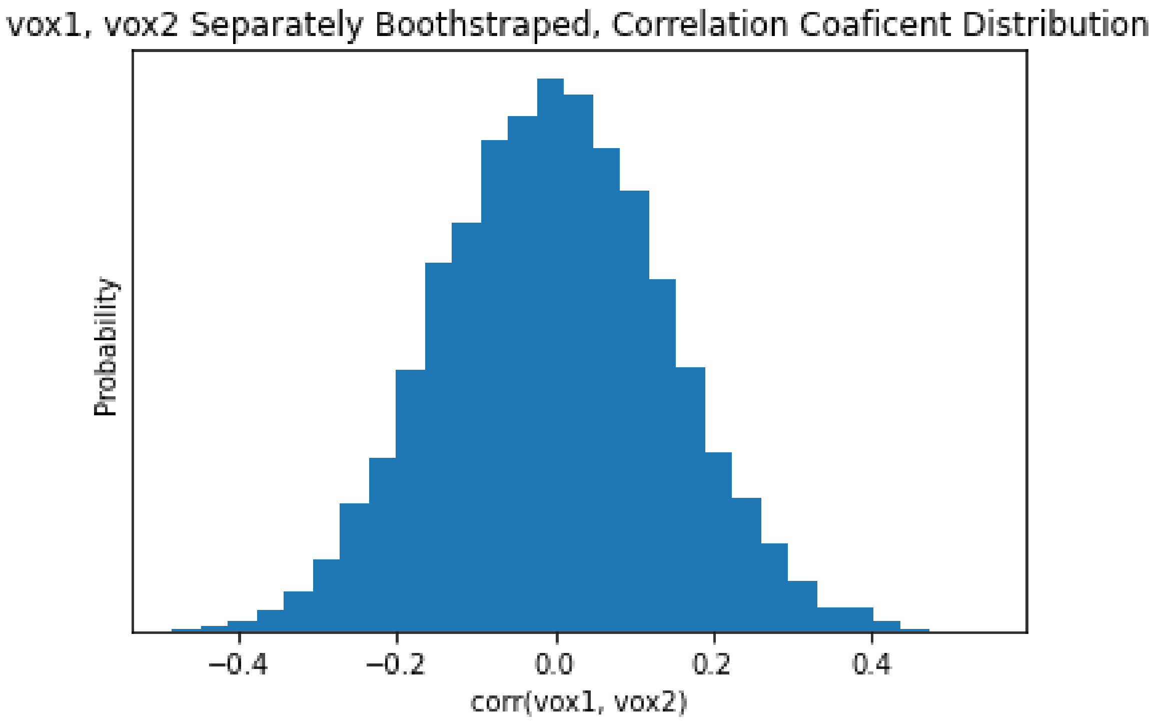
$$p = P\left(Z > \frac{k - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 1 - P\left(Z < \frac{k - E[A]}{\sqrt{var(A)}}\right) \quad (7)$$

the code for this is given below.

```
    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    vox1 = data2["vox1"]
    vox2 = data2["vox2"]

    # null hypothesis is they have 0 correlation. so we will bootstrape the 2
#sets seperatly this way if they have any corelation it will be broken, then we
#will look into its distribuiton and find the probability of getting the
#corelation etween the given vox1 and vox2

    bootstraps_vox1 = bootstrap_singel(10000, vox1.transpose()) # creating 10000
# bootstraps for vox1 and vox2 seperatly. they are row vectors and most finctions
# are defiened for collum vector so we take their transpose.
    bootstraps_vox2 = bootstrap_singel(10000, vox2.transpose())

    bootstraps_vox1_vox2_correlation = np.zeros(10000) # we will fill this in

    for i in range(10000): # Computing the correlation coaficent for 10000
#bootstrap vox1, vox2 pairs.

        bootstraps_vox1_vox2_correlation[i] = (np.corrcoef((bootstraps_vox1[i,:])
.transpose(), (bootstraps_vox2[i,:]).transpose() ))[0, 1] # computing the
#correlation matrix for the ith bootstraps vox1 vox2 pair.

    plt.figure() # I wanted to show how the distribution looked
    plt.title('vox1, vox2 Separately Boothstraped, Correlation Coaficent 
Distribution')
    plt.xlabel('corr(vox1, vox2)')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstraps_vox1_vox2_correlation, bins=29, density=True)

    # now we will compute the 1 tailed p value of getting the correlation of the 
#origanal vox1, vox2 given to us.

    given_correlation = (np.corrcoef(vox1, vox2))[0, 1] # computing the 
#correlation of the origanal vox1, vox2 given to us.

    p = 1 - norm.cdf(np.abs( (given_correlation - np.mean(bootstraps_vox1_vox2_co
rrelation)) / (np.std(bootstraps_vox1_vox2_correlation)) )) # here we compute 
#P(|bootstrap_mean_difrence| > |given_mean_difrence|) by using the CDF of the 
#standart normla distribution.

    significant_points = np.where(bootstraps_vox1_vox2_correlation > given_correl
ation) # finding the significant points

    one_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 1 t
ailed p value

    print("One-tailed p-
value for the null hypothesis that two voxel responses have zero correlation is "
 + str(one_tailed_p_value) + " for counting bootstrap results over the given corr
elation.\nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and t
he 2 populations have positive correlaiton.\n")


    print("This makes sense because if we look at the result of part b the 
probability of getting a zero correlation was 0% and all the correlation 
coaficent we found were positive so the probability of these voxes having 0 or 
lower correlation being smaller then 5% is very logical.")
```

this code gives the output plot shown below.



It also finds one-tailed, p-value as 0.0. This means our null-hypothesis is not correct and vox1, vox2 have a positive correlation. This is something which we were expecting, in part b, distribution of bootstraps of the given vox1, vox2 showed us 0% of the results had 0 correlation also, if we look at the plot in part b the probability of getting a 0 correlation is 0, and all the correlation value distribution is on the positive right side. In this part we computed if vox1 and vox2 had 0 correlation the probability of getting a correlation equal to or higher than the correlation between the original vox1, vox2 is 0.00351% which is very logical. Like the question said, estimation of confidence intervals and hypothesis testing are dual problems. In part b and c we computed and showed the same thing in different ways. In part b we showed trough the confidence interval of the original vox1 and vox2, it was almost impossible for vox1 and vox2 to have 0 correlation. In part c, using the null-hypotheses; vox1 and vox2 have 0 correlation we computed the probability to get the correlation value the original vox1, vox2 had and saw it was extremely small. These are showing us the same results through different computations.

I also found the one-tailed p-value as 0.0000351 by first fitting a Gaussian to the null-hypothesis distribution and the solving equation 7. The 2 p-values have similar results because the distribution acts very much like a Gaussian.

What we did in part b, c of question 1 is an example of what we did in part b of this question.

d) The average BOLD responses in a face-selective region of the human brain have been recorded in two separate experiments. The responses of this region to building images (1st experiment) and face images (2nd experiment) are stored in the variables building and face for 20 subjects. Assume that the same subject population was recruited in both experiments. Use bootstrapping (10000 iterations) to calculate the two-tailed p-value for the null hypothesis that there is no difference between the building and face responses.

This question is similar to part b. If the same population took part in both experiments, we need to sample the 2 experiments responses identically. But we cannot simply re-use the bootstrap function which we used in part b. The null-hypothesis is faces and buildings create the same response. This means a participant in the experiment can give the response he gave to a face to a building or the response he gave to a building to a face. So, I need to create a bootstrap function that chooses which people will participate in both experiments and what response they will give to each of the experiments.

To calculate the two-tailed p-value I decided to use the difference of means of the two experiments responses. I need to find the distribution of difference of means of the two experiments, then I could find the probability of getting a mean difference equal or greater than the given original experiments mean difference to calculate the p-value. To do this I counted the number of bootstraps that fall in the rejection region and divided that number by the number of the total number of both straps. I also computed the p-value by first fitting a Gaussian to the null-hypothesis distribution and then solving equation 8.

$$A = mean\ difference\ distribution\ of\ building\ , face\ responses\ for\ null\ hypothesis$$

$$k = E[building\ responses] - E[face\ responses]$$

$$p = 2 * P(A > |k|)$$

$$p = 2 * P\left(\frac{A - E[A]}{\sqrt{var(A)}} > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 2 * P\left(Z > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 2 * \left(1 - P\left(Z < \frac{|k| - E[A]}{\sqrt{var(A)}}\right)\right) \tag{8}$$

Now I need to do bootstrapping to find the distribution of the mean differences for the null-hypothesis. For this I created a new bootstrapping function named bootstrap_same_polulation for this question. It generates the desired number of bootstraps for

2 sets simultaneously for the null-hypothesis of this part. It does this by first choosing which participants will be part of an iteration. Then for each participant, it randomly chooses which response it will give to the first experiment, then it randomly chooses which response it will give to the second experiment. If a participant is chosen more than once for an iteration, then the previous step is repeated for each time a participant is chosen in that iteration. This way, it generates 20 responses for both of the experiments and finishes one bootstrap iteration. It repeats this process until it generates the desired number of iterations. In the end, it returns a face experiment responses matrix and a building experiment responses matrix. The code for this function is given below.

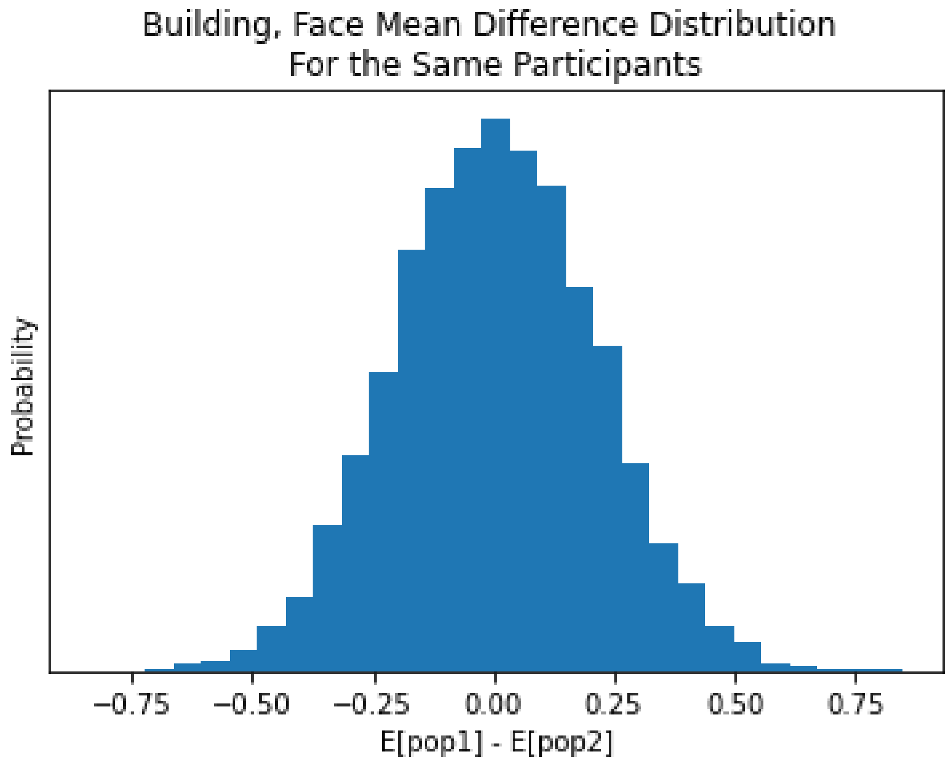
```python
def bootstrap_same_polulation(iteration_count, x, y): # the function to
#callculate n bootstraps for 2 vectors asumming the 2 data came from the sane
#popolation an the responses have no difference
    # I assume the first response for both data came from the same participant
#the second response for both data came from the same participant and so on.

    building_bootstraps = []
    face_bootstraps = []

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x))
# chosing the index of the values which will be added to this iteration, if we
#decide the ith response will be part of the boothstrap. Then it means we need to
# add the ith response of either one of the inputs to both of the outputs.

        building_bootstrap = []
        face_bootstrap = []


        for j in bootstrap_indexs: # for each participant we will decide which
#reponses to add the the 2 outputs

            random_bits = np.random.randint(2, size = 2) # we randomly choose
#which response will be added here

            if bool(random_bits[0]):

                building_bootstrap.append(x[j]) # adding the jth response of x
#with 50% probability

            else:
                building_bootstrap.append(y[j]) # adding the jth response of y
#with 50% probability

            if bool(random_bits[1]):

                face_bootstrap.append(x[j]) # adding the jth response of x with
#50% probability

            else:

                face_bootstrap.append(y[j]) # adding the jth response of y with
#50% probability


        building_bootstraps.append(building_bootstrap) # adding the bootstrap
#iteration to the output list for building

        face_bootstraps.append(face_bootstrap) # adding the bootstrap iteration
#to the output list for y

    return np.array(building_bootstraps), np.array(face_bootstraps)
```

Now that I have the bootstrap function, I put the given faces responses and building responses into the new bootstrap function for 10000 iterations. Then I calculated the sample mean of each building experiment and face experiment. Then subtracted their means from each other to find 10000 mean differences for the 10000 iterations. These give the distribution of mean differences of the two experiments for the null-hypothesis. I plotted this distribution to show how it looks. Then I counted the number of points which fell in the rejection region in the dsitribution, and divided that number with the number of bootstraps to find the two-tailed p-value. I also used the distribution to solve equation 8 and found the two-tailed p-value that way. The code which does this is given below.

```
def question2_part_d():

    print("\nAnswer of question 2 part d: \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    building = data2["building"]
    face = data2["face"]

    # null hypothesis is they have no difference in their response. If they have
no difference in their response then their difrence in means will be zero. So we
will compute the difference in means distribution asuming they have no difrence i
```

```
n their responses. then I will find the probability of obtaining the same difrenc
e in means as the original given data.

    boothstrap_building, boothstrap_face = bootstrap_same_polulation(10000, build
ing.transpose(), face.transpose())

    bootstrap_building_mean = np.mean(boothstrap_building, axis=1) # calculating
mean of building for each of the 10000 bootstraps
    bootstrap_face_mean = np.mean(boothstrap_face, axis=1) # calculating mean of
face for each of the 10000 bootstraps

    bootstrap_mean_difrence = bootstrap_building_mean - bootstrap_face_mean # cal
culating mean difrence for each of the 10000 bootstraps, this form the mean difre
nce distribution of the population.

    plt.figure() # I wanted to show how it looks
    plt.title('Building, Face Mean Difference Distribution \nFor the Same
Participants')

    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difrence, bins=29, density=True)

    given_mean_difrence = np.mean(building)-
np.mean(face) # computing the mean difrence for the origanal pop1, pop2 given to
us

    p = 2 * (1 - norm.cdf(np.abs( (given_mean_difrence - np.mean(bootstrap_mean_d
ifrence)) / (np.std(bootstrap_mean_difrence)) ))) # here we compute P(|bootstrap_
mean_difrence| > |given_mean_difrence|) by using the CDF of the standart normla d
istribution.


    significant_points = np.where(np.abs(bootstrap_mean_difrence) > np.abs(
given_mean_difrence)) # finding the significant points

    two_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 2
#tailed p value


    print("Two-tailed p-
value for the null hypothesis that building and face create the same response is
" + str(two_tailed_p_value) + " for counting bootstrap results over the given dif
ference. \nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and b
uildings and faces create different responses.")
```

This code plots the graph shown below.



It finds the two-tailed p-value as 0.0004. This means the null hypothesis is highly unlikely, so buildings and faces generally create a different BOLD response in a face-selective region of the human brain.

It also found the two-tailed p-value as 0.0000358 by first fitting a Gaussian to the null-hypothesis distribution and the solving equation 8. The 2 p-values have similar results because the distribution acts very much like a Gaussian, because the means are computed from sets which are composed of 20 elements.

e) Repeat the exercise in part d, but this time assuming that the subject populations recruited for the two experiments are distinct. Use bootstrapping (10000 iterations) to calculate the two-tailed p-value for the null hypothesis that there is no difference between the building and face responses

The situation in this problem is almost identical to one given in part a of this question. Null-hypothesis is, the two stimuli, face and a building have no different responses. Another way to say the same thing is, the experiment's results follow the same distribution. Then we can compute p-value from by first finding the distribution of mean differences of building experiment and face experiment, then finding the probability of getting the mean difference of the originally given experiment results. To find the probability once again I counted the number of bootstraps that fall in the rejection region and then divided that number by the total bootstrap count. Again I also computed the p-value by first fitting a Gaussian to the null-hypothesis distribution and then doing the computation given below.

$$A = mean\ difference\ distribution\ of\ building\ , face\ responses\ for\ null\ hypothesis$$

$$k = E[building\ responses] - E[face\ responses]$$

$$p = 2 * P(A > |k|)$$

$$p = 2 * P\left(\frac{A - E[A]}{\sqrt{var(A)}} > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

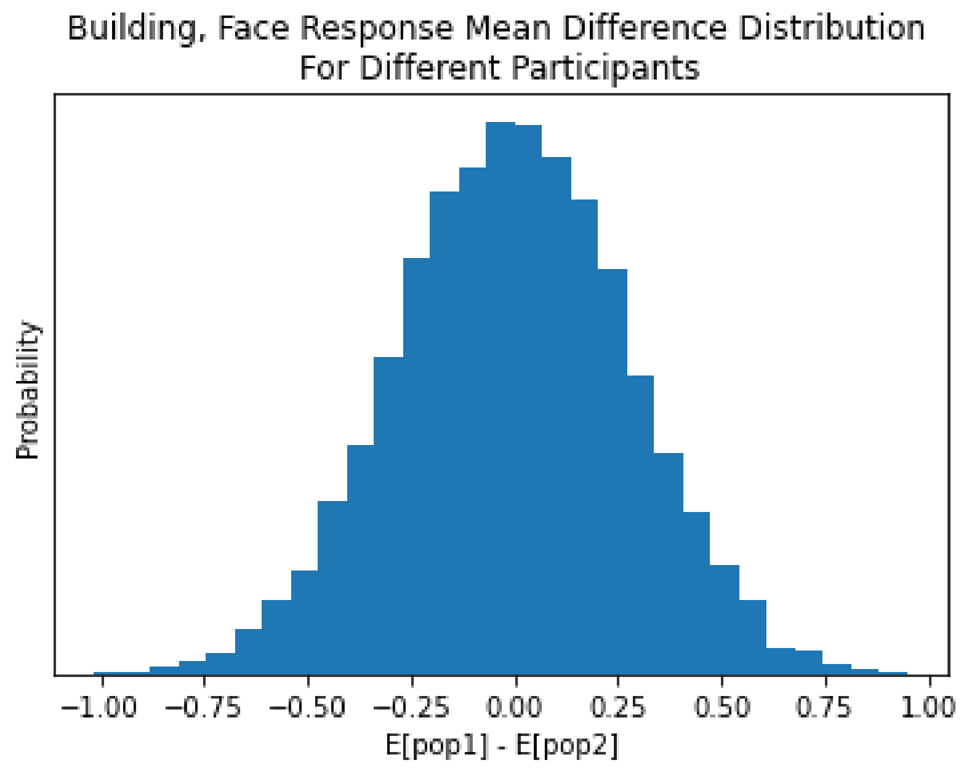
$$p = 2 * P\left(Z > \frac{|k| - E[A]}{\sqrt{var(A)}}\right)$$

$$p = 2 * \left(1 - P\left(Z < \frac{|k| - E[A]}{\sqrt{var(A)}}\right)\right) \tag{9}$$

To find the distribution of building ,face responses for null hypothesis I will do what I did for the part a. I concatenated building responses and face responses and used it as the input of the bootstrap_singel with an iteration count of 10000. Then I separated the first 20 elements of all 10000 iterations as a building responses matrix and the last 20 elements of all 10000 iterations as a face responses matrix. Then I found the building responses and face responses mean for all 10000 iterations. Then subtracted face responses means from building responses means to calculate the mean difference of all the iterations. These mean differences give the distribution of the null hypothesis. I plotted the null hypothesis distribution to show how it looks like. I counted the number of bootstraps that fall in the rejection region and then divided that number by the total bootstrap count to find the two-tailed p-value. I also computed the distributions sample mean and standard deviation. Then, using equation 9 I computed the two-tailed p-value again. The code for this is given below.

```
    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    building = data2["building"]
    face = data2["face"]

    # null hypothesis is they have no difference in their response. If they have
#no difference in their response then their difrence in means will be zero. So we
# will compute the difference in means distribution asuming they have no difrence
# in their responses. then I will find the probability of obtaining the same
#difrence in means as the original given data.
    # what we do in this question is same as part a.

    building_face = np.concatenate((building, face), axis = 1) # the neron group
#from our null hypotezis

    bootstraps = bootstrap_singel(10000, building_face.transpose()) # creating
#10000  bootstraps from the total population

    bootstrap_building = bootstraps[:,0:20] # picking the first 20 elements as
#the building for each of the 10000 bootstraps
    bootstrap_face = bootstraps[:,20:40] # picking the last 20 elements as the
#face for each of the 10000 bootstraps

    bootstrap_building_mean = np.mean(bootstrap_building, axis=1) # calculating
#mean of building experiment for each of the 10000 bootstraps

    bootstrap_face_mean = np.mean(bootstrap_face, axis=1) # calculating mean of
#face experiment for each of the 10000 bootstraps


    bootstrap_mean_difference = bootstrap_building_mean - bootstrap_face_mean
# calculating mean difrence for each of the 10000 bootstraps, this form the mean
#difrence distribution of the population.



    given_mean_difrence = np.mean(building)-
np.mean(face) # computing the mean difrence for the origanal building, face given
# to us

    p = 2 * (1 - norm.cdf(( ( np.abs(given_mean_difrence) - np.mean(bootstrap_mea
n_difference) ) / (np.std(bootstrap_mean_difference)) ))) # here we compute
#P(|bootstrap_mean_difrence| > |given_mean_difrence|) by using the CDF of the
#standart normla distribution.



    significant_points = np.where(np.abs(bootstrap_mean_difference) > np.abs(give
n_mean_difrence)) # finding the significant points

    two_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 2
#tailed p value

    print("Two-tailed p-
value for the null hypothesis that building and face create the same response is
" + str(two_tailed_p_value) + " for counting bootstrap results over the given dif
ference. \nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and b
uildings and faces create different responses.")

    print("\nThis means there is a difrence response for building and face,
regardless of our assumption of the participitants.")

    plt.figure() # I wanted to show how it looks
    plt.title(Building, Face Response Mean Difference Distribution \nFor
Different Participants')

    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difference, bins=29, density=True)
```

This gives the output plot of.



It also finds the two-tailed p-value as 0.0065. This is smaller then 0.05 so the null hypothesis is incorrect and faces and buildings generate different responses in a face-selective region of the human brain. Also if we look at this result and the result from part d we can say the null-hypothesis is incorrect regardless of if the 2 experiments are done with the same population.

It also found the two-tailed p-value as 0.00699 by first fitting a Gaussian to the null-hypothesis distribution and the solving equation 8. The 2 p-values have similar results because the distribution acts very much like a Gaussian, because the means are computed from sets which are composed of 20 elements.

CODE:

As a formality I am adding all of my code as it is at the end of the report:

```
# -*- coding: utf-8 -*-

import sys # the primer already has it

import numpy as np # this brings simple matrix operations to python
import matplotlib.pyplot as plt # this brings nice loking plots to python

from scipy.stats import norm # used to calculet CDF's used to calculate the p val
ue
import scipy.io as sio # this is used to load MATLAB files

"""
If you want to run this code in a IDE you need to commend out the line below and
write the line
question = '1'
then run the code for question 1 and write the line
question = '2'
then run the code for question 2
"""

question = sys.argv[1]

def Batu_Arda_Düzgün_21802633_hw3(question):
    if question == '1' :
        print("Answer of question 1.) \n")
        question1_part_a()
        question1_part_b()
        question1_part_c()

    elif question == '2' :
        print("Answer of question 2.) \n")
        question2_part_a()
        question2_part_b()
        question2_part_c()
        question2_part_d()
        question2_part_e()




def ridge_regression(X, y, lamda):
    return np.linalg.inv(X.T.dot(X) + lamda * np.identity(np.shape(X)[1])).dot(X.
T).dot(y) # matrix operation which gives the weight vector for the ridge regesion
.



def cross_validationation(X, y, K, lamda):

    fold_size = int(np.size(y)/K) # size of test and validationation


    validation_R2 = 0 # different R^2 will be summed up in here and then will be
devided by K to find the average
    test_R2 = 0

    for i in range(K):

        train_data_ind, test_data_ind, validation_data_ind = [], [], []

        test_data_start = i * fold_size # stating point of validation set for the
 given K
        validation_data_start = (i+1) * fold_size # stating point of test set for
 the given K
        train_data_start = (i+2) * fold_size # stating point of training set for
the given K

        for j in range(2 * np.size(y)): # deciding which set every index in (0 ,
999) will go to.

            if j in range(test_data_start, validation_data_start):

                test_data_ind.append(j % np.size(y))

            if j in range(validation_data_start, train_data_start):

                validation_data_ind.append(j % np.size(y))
            if  j in range(train_data_start, test_data_start + np.size(y)):

                train_data_ind.append(j % np.size(y))


        x_validation, x_test, x_training = X[validation_data_ind], X[test_data_in
d], X[train_data_ind] # palcing every input output pare in their respective set

        y_validation, y_test, y_training = y[validation_data_ind], y[test_data_in
d], y[train_data_ind]


        validation_weight = ridge_regression(x_training , y_training , lamda) # r
iged rigresion to find the weight of the input parameters

        validation_R2 += (np.corrcoef(y_validation.transpose(), (x_validation.dot
(validation_weight)).transpose() )[0, 1]) ** 2 # corrcoef returns the coralation 
coaficent matrix of its inputs so here its outpıts [0 1] is the coralation coafic
ent coaficent betwen theh real y and the y we calculated form the fit model. we t
akes its square to calculate the R^2 of the model


        test_weight = ridge_regression(np.concatenate((x_validation, x_training)
, axis=0) , np.concatenate((y_validation, y_training) , axis=0) , lamda) # riged 
rigresion to find the weight of the input parameters for both the traing and the 
validation

        test_R2 += (np.corrcoef( y_test.transpose() , (x_test.dot(test_weight)).t
ranspose() )[0, 1]) ** 2 # again finding R^2 for the test.


    validation_R2 /= K # taking the R^2 average
    test_R2 /= K

    return validation_R2, test_R2




def question1_part_a():

    print("\nAnswer of question 1 part a: \n")
    print("every time this code is run part b and part c of this code will give a
 Slightly different output because of the randomness in bootstraping \n ")
```

```
    print("this will take some time to compute because I am testing for 1250 diff
erent lambda values \n")

    data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
    Xn = data["Xn"]
    Yn = data["Yn"]

    lamda_values = np.logspace(-
3, 12, num=1250, base=10) # different lambda values to calculate proportion of ex
plained variance

    validation_R2_for_lamda_values = [] # we will fill these with the R^2 values
for all the lambda
    test_R2_for_lamda_values = []


    for lamda in lamda_values: # doing the K fold regresion for each of the lambd
a values

        validation_R2 , test_R2 = cross_validationation(Xn, Yn, 10, lamda)

        validation_R2_for_lamda_values.append(validation_R2)
        test_R2_for_lamda_values.append(test_R2)


    plt.figure(figsize=(12, 8)) # plotting R^2 curves
    plt.plot(lamda_values, test_R2_for_lamda_values)
    plt.plot(lamda_values, validation_R2_for_lamda_values)
    plt.legend(['Test', 'Validation',])
    plt.ylabel('proportion of explained variance R^2')
    plt.xlabel('ridge parameter lambda')
    plt.title('R^2 vs lambda')
    plt.xscale('log')
    plt.grid()


    max_R2 =  max(validation_R2_for_lamda_values)

    lamda_opt_index = validation_R2_for_lamda_values.index(max_R2)

    lamda_opt = lamda_values[lamda_opt_index]

    performance_of_lambda_opt = test_R2_for_lamda_values[lamda_opt_index] # findi
ng the model performance for the optimum lambda
```

```
    print('optimal ridge parameter across cross-
validation folds, measured on the validation set =' + str(lamda_opt) + " and the
model with this lambda, gives a model performance of R^2 = " + str(performance_of
_lambda_opt))



def bootstrap(iteration_count, X, y, lamda): # the function to generate input wei
ghts for n bootstraps for a given input, output and ridged regreasion

    weights = [] # we will fill this in

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.size(y), np.size(y)) # chosing the
 index of the input output pairs which will be added to this iteration

        X_iteration = X[bootstrap_indexs]  # adding the choosen input output pair
s in to the iteration
        y_iteration = y[bootstrap_indexs]

        weights.append(ridge_regression(X_iteration, y_iteration, lamda)) # addin
g the weights for this iteration to the output list

    return weights



def question1_part_b():

    print("\nAnswer of question 1 part b: \n")


    data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
    Xn = data["Xn"]
    Yn = data["Yn"]

    weights = np.array(bootstrap(500, Xn, Yn, 0)) # getting the weights form 500
bootstrap iterations

    plt.figure(figsize=(8, 4)) # we know this distribution should look like a gau
sian but I wanted to show it looks like a gausian too.
    plt.title('the first weight parameters distribution for lambda = 0')
    plt.xlabel('w_1')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,0], bins=19, density=True)

    plt.figure(figsize=(8, 4)) # we know this distribution should look like a gau
sian but I wanted to show it looks like a gausian too.
    plt.title('the 4th weight parameters distribution for lambda = 0')
    plt.xlabel('w_4')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,3], bins=19, density=True)

    plt.figure(figsize=(8, 4)) # we know this distribution should look like a gau
sian but I wanted to show it looks like a gausian too.
    plt.title('the 8th weight parameters distribution for lambda = 0')
    plt.xlabel('w_8')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(weights[:,7], bins=19, density=True)


    weights_mean = np.mean(weights, axis=0) # calculating the mean and standart d
eviation of each of the 100 weights
    weights_std = np.std(weights, axis=0)


    _95_percent_confidence = 2 * np.concatenate((weights_std.transpose(),weights_
std.transpose()) , axis = 0) # the weights are normal distribution so 2 standart 
deviations to the both sides equals 95% confidence interval.


    p_values = 2 * (1 - norm.cdf(np.abs(weights_mean/weights_std))) # 2 tailed p 
value for the weight to have 0 mean


    significant_weights = np.where(p_values < 0.05) # finding the significant p v
alues

    significant_weights = significant_weights[0]

    significant_weights_mean = weights_mean[significant_weights] # will be used i
n the plot


    plt.figure(figsize=(12, 8)) # plot of the weights and their 95% confidence in
tervals
```

```
    plt.grid() # grids so we can see the significant points are points which do n
ot have a confidance interval which intercept 0
    plt.errorbar(np.arange(0,100), weights_mean , yerr = _95_percent_confidence ,
 ecolor='b', fmt='ok', capsize=3) # plotting all the weights
    plt.errorbar(significant_weights, significant_weights_mean, fmt='or') # marki
ng the significant weights with red.
    plt.ylabel('Weight Values with 95% confidence intervals')
    plt.xlabel('Weight Indexes')
    plt.title('Ridge Regression for lambda = 0 with %95 confidence interval')

    print('I marked the significant weights with red')



def question1_part_c():

    print("\nAnswer of question 1 part c: \n")

    data = sio.loadmat('matlab_HW3_data2') # loading the data given to us
    Xn = data["Xn"]
    Yn = data["Yn"]

    opt_lamda = 353.5 # I wrote this value by hand here because its computation t
akes a very long time and re calculating it here would be excessive

    weights = np.array(bootstrap(500, Xn, Yn, opt_lamda)) # getting the weights f
orm 500 bootstrap iterations

    weights_mean = np.mean(weights, axis=0) # calculating the mean and standart d
eviation of each of the 100 weights
    weights_std = np.std(weights, axis=0)


    _95_percent_confidence = 2 * np.concatenate((weights_std.transpose(),weights_
std.transpose()) , axis = 0) # the weights are normal distribution so 2 standart
deviations to the both sides equals 95% confidence interval.



    p_values = 2 * (1 - norm.cdf(np.abs(weights_mean/weights_std))) # 2 tailed p
value for the weight to have 0 mean



    significant_weights = np.where(p_values < 0.05) # finding the significant p v
alues
```

```
    significant_weights = significant_weights[0]

    significant_weights_mean = weights_mean[significant_weights] # will be used i
n the plot

    print('I marked the significant weights with red')

    print('riged regresion has more significnat weights. Normaly we would expect
the riged regresion to give out less number of significant parameters but here it
 also decresased the variance of the parameters which caused most of the weights
which are on the edge of becoming significant signifiacant. Because of this riged
 regreasion version has more significant weights.')

    plt.figure(figsize=(12, 8)) # plot of the weights and their 95% confidence in
tervals
    plt.grid() # grids so we can see the significant points are points which do n
ot have a confidance interval which intercept 0
    plt.errorbar(np.arange(0,100), weights_mean , yerr = _95_percent_confidence ,
 ecolor='b', fmt='ok', capsize=3) # plotting all the weights
    plt.errorbar(significant_weights, significant_weights_mean, fmt='or') # marki
ng the significant weights with red.
    plt.ylabel('Weight Values with 95% confidence intervals')
    plt.xlabel('Weight Indexes')
    plt.title('Ridge Regression for lambda = ' + str(opt_lamda) + ' with %95 conf
idence interval')
    plt.show() # this privents the plot from closing themselves at the end





def bootstrap_singel(iteration_count, x): # the function to callculate n bootstra
ps of a vector

    x_bootstraps = []

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x))  #
 chosing the index of the values which will be added to this iteration

        x_bootstrap = x[bootstrap_indexs] # adding the choosen values in to the i
teration
```

```
        x_bootstraps.append(x_bootstrap) # adding the bootstrap iteration to the 
output list

    return np.array(x_bootstraps)



def question2_part_a():

    print("\nAnswer of question 2 part a: \n")

    print("every time this code is run part a, part b, part c, part d and part e 
of this code will give a Slightly different output because of the randomness in b
ootstraping \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    pop1 = data2["pop1"]
    pop2 = data2["pop2"]

    # null hypothesis is they are from the same distribution. so we will add the 
2 sets and create bootstraps from that and then we will take the first 7 as the v
alues in vox1 and the last 5 values as vox2. then we will compare their difrence 
in means and look at how probable it is to get the mean difrence we get in this p
articular case.

    pop1_pop2 = np.concatenate((pop1, pop2), axis = 1) # the neron group from our
 null hypotezis

    bootstraps = bootstrap_singel(10000, pop1_pop2.transpose()) # creating 10000
 bootstraps from the total population

    bootstrap_pop1 = bootstraps[:,0:7] # picking the first 7 elements as the pop1
 for each of the 10000 bootstraps
    bootstrap_pop2 = bootstraps[:,7:12] # picking the last 5 elements as the pop2
 for each of the 10000 bootstraps

    bootstrap_pop1_mean = np.mean(bootstrap_pop1, axis=1) # calculating mean of p
op1 for each of the 10000 bootstraps
    bootstrap_pop2_mean = np.mean(bootstrap_pop2, axis=1) # calculating mean of p
op2 for each of the 10000 bootstraps

    bootstrap_mean_difrence = bootstrap_pop1_mean - bootstrap_pop2_mean # calcula
ting mean difrence for each of the 10000 bootstraps, this form the mean difrence 
distribution of the population.
```

```
    plt.figure() # I wanted to show it looks like a gausian
    plt.title('pop1, pop2 Mean Difference Distribution')
    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difrence, bins=29, density=True)

    # now we will compute the 2 tailed p value of getting the mean difrence for t
he origanal pop1, pop2 given to us.

    given_mean_difrence = np.mean(pop1)-
np.mean(pop2) # computing the mean difrence for the origanal pop1, pop2 given to 
us

    p = 2 * (1 - norm.cdf( ( np.abs(given_mean_difrence) - np.mean(bootstrap_mean
_difrence)) / (np.std(bootstrap_mean_difrence)) )) # here we compute P(|bootstrap
_mean_difrence| > |given_mean_difrence|) by using the CDF of the standart normla 
distribution.

    significant_points = np.where(np.abs(bootstrap_mean_difrence) > np.abs(given_
mean_difrence)) # finding the significant points

    two_tailed_p_value = np.size(significant_points[0])/10000 # calcaulatin 2 tai
led p value

    print("Two-tailed p-
value for the null hypothesis that the two datasets follow the same distribution 
is " + str(two_tailed_p_value) + " for counting bootstrap results over the given 
difference.\nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and t
he 2 populations follow different distributions.")



def bootstrap_double(iteration_count, x, y): # the function to callculate n boots
traps for 2 vectors with out breaking the coralation between them.

    x_bootstraps = []
    y_bootstraps = []


    for i in range(iteration_count):
        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x)) #
chosing the index of the values which will be added to this iteration

        x_bootstrap = x[bootstrap_indexs] # adding the choosen values in to the i
teration for x
        x_bootstraps.append(x_bootstrap) # adding the bootstrap iteration to the
output list for x

        y_bootstrap = y[bootstrap_indexs] # adding the choosen values in to the i
teration for y
        y_bootstraps.append(y_bootstrap) # adding the bootstrap iteration to the
output list for y

    return np.array(x_bootstraps), np.array(y_bootstraps)



def question2_part_b():

    print("\nAnswer of question 2 part b: \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    vox1 = data2["vox1"]
    vox2 = data2["vox2"]

    bootstraps_vox1, bootstraps_vox2 = bootstrap_double(10000, vox1.transpose(),
vox2.transpose()) # creating 10000  bootstraps for vox1, vox2 pairs. they are row
 vectors and most fınctions are defiened for collum vector so we take their trans
pose.

    bootstraps_vox1_vox2_correlation = np.zeros(10000) # we will fill this in

    for i in range(10000): # Computing the correlation coaficent for 10000 bootst
rap vox1, vox2 pairs.

        bootstraps_vox1_vox2_correlation[i] = (np.corrcoef((bootstraps_vox1[i,:])
.transpose(), (bootstraps_vox2[i,:]).transpose() ))[0, 1] # comuting the correlat
ion matrix for the ith bootstraps vox1 vox2 pair.

    plt.figure() # I wanted to show how the distribution looked
    plt.title('vox1, vox2 Correlation Coaficent Distribution')
    plt.xlabel('corr(vox1, vox2)')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstraps_vox1_vox2_correlation, bins=29, density=True)
    bootstraps_vox1_vox2_correlation_mean = np.mean(bootstraps_vox1_vox2_correlat
ion) # mean of the correlation

    # can use std to calculate because the distribution is not gausian.

    bootstraps_vox1_vox2_correlation_sorted = np.sort(bootstraps_vox1_vox2_correl
ation) # we order the correlation the pick the 250th and 9750th term because they
 should be on the 2 borders of 95% interval.

    lower_95_percent_confidence_border = bootstraps_vox1_vox2_correlation_sorted[
249]
    upper_95_percent_confidence_border = bootstraps_vox1_vox2_correlation_sorted[
9749]

    zero_correlation_points = np.where(bootstraps_vox1_vox2_correlation_sorted <
0.05) # the correlation values are floting point so if we where to look at exeact
ly 0 we woulden't get any matches

    percenteg_of_zero_points = np.size(zero_correlation_points[0]) * 100 / 10000
# turning the count to a percentage.

    print("Mean of the correlation coefficient distribution = " + str(bootstraps_
vox1_vox2_correlation_mean))
    print("95% confidence interval of the correlation coefficient distribution =
(" + str(lower_95_percent_confidence_border) + ", " + str(upper_95_percent_confid
ence_border) + ")")
    print("Percentile of bootstrap distribution, corresponding to a correlation v
alue of 0 = " + str(percenteg_of_zero_points) + "%")



def question2_part_c():

    print("\nAnswer of question 2 part c: \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    vox1 = data2["vox1"]
    vox2 = data2["vox2"]

    # null hypothesis is they have 0 correlation. so we will bootstrape the 2 set
s seperatly this way if they have any corelation it will be broken, then we will
look into its distribuiton and find the probability of getting the corelation etw
een the given vox1 and vox2
```

```
    bootstraps_vox1 = bootstrap_singel(10000, vox1.transpose()) # creating 10000
 bootstraps for vox1 and vox2 seperatly. they are row vectors and most fınctions
are defiened for collum vector so we take their transpose.
    bootstraps_vox2 = bootstrap_singel(10000, vox2.transpose())

    bootstraps_vox1_vox2_correlation = np.zeros(10000) # we will fill this in

    for i in range(10000): # Computing the correlation coaficent for 10000 bootst
rap vox1, vox2 pairs.

        bootstraps_vox1_vox2_correlation[i] = (np.corrcoef((bootstraps_vox1[i,:])
.transpose(), (bootstraps_vox2[i,:]).transpose() ))[0, 1] # comuting the correlat
ion matrix for the ith bootstraps vox1 vox2 pair.

    plt.figure() # I wanted to show how the distribution looked
    plt.title('vox1, vox2 Separately Boothstraped, Correlation Coaficent Distribu
tion')
    plt.xlabel('corr(vox1, vox2)')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstraps_vox1_vox2_correlation, bins=29, density=True)

    # now we will compute the 1 tailed p value of getting the correlation of the
origanal vox1, vox2 given to us.

    given_correlation = (np.corrcoef(vox1, vox2))[0, 1] # computing the correlati
on of the origanal vox1, vox2 given to us.

    p = 1 - norm.cdf(( (np.abs(given_correlation) - np.mean(bootstraps_vox1_vox2_
correlation)) / (np.std(bootstraps_vox1_vox2_correlation)) )) # here we compute P
(|bootstrap_mean_difrence| > |given_mean_difrence|) by using the CDF of the stand
art normla distribution.

    significant_points = np.where(bootstraps_vox1_vox2_correlation > given_correl
ation) # finding the significant points

    one_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 1 t
ailed p value

    print("One-tailed p-
value for the null hypothesis that two voxel responses have zero correlation is "
 + str(one_tailed_p_value) + " for counting bootstrap results over the given corr
elation.\nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and t
he 2 populations have positive correlaiton.\n")

    print("This makes sense because if we look at the result of part b the probab
ility of getting a zero correlation was 0% and all the correlation coaficent we f
ound were positive so the probability of these voxes having 0 or lower correlatio
n being smaller then 0.005% is very logical.")



def bootstrap_same_polulation(iteration_count, x, y): # the function to callculat
e n bootstraps for 2 vectors asumming the 2 data came from the sane popolation an
 the responses have no difrence
    # I assume the first response for both data came from the same participant th
e second response for both data came from the same participant and so on.

    building_bootstraps = []
    face_bootstraps = []

    for i in range(iteration_count):

        bootstrap_indexs = np.random.choice(np.arange(np.size(x)), np.size(x)) #
chosing the index of the values which will be added to this iteration, if we deci
de the ith response will be part of the boothstrap. Then it means we need to add
the ith response of either one of the inputs to both of the outputs.

        building_bootstrap = []
        face_bootstrap = []


        for j in bootstrap_indexs: # for each participant we will decide which re
ponses to add the the 2 outputs

            random_bits = np.random.randint(2, size = 2) # we randomly choose whi
ch response will be added here

            if bool(random_bits[0]):

                building_bootstrap.append(x[j]) # adding the jth response of x wi
th 50% probability

            else:

                building_bootstrap.append(y[j]) # adding the jth response of y wi
th 50% probability
```

```
            if bool(random_bits[1]):

                face_bootstrap.append(x[j]) # adding the jth response of x with 5
0% probability

            else:

                face_bootstrap.append(y[j]) # adding the jth response of y with 5
0% probability


        building_bootstraps.append(building_bootstrap) # adding the bootstrap ite
ration to the output list for building

        face_bootstraps.append(face_bootstrap) # adding the bootstrap iteration t
o the output list for y


    return np.array(building_bootstraps), np.array(face_bootstraps)



def question2_part_d():

    print("\nAnswer of question 2 part d: \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    building = data2["building"]
    face = data2["face"]

    # null hypothesis is they have no difference in their response. If they have
no difference in their response then their difrence in means will be zero. So we
will compute the difference in means distribution asuming they have no difrence i
n their responses. then I will find the probability of obtaining the same difrenc
e in means as the original given data.

    boothstrap_building, boothstrap_face = bootstrap_same_polulation(10000, build
ing.transpose(), face.transpose())

    bootstrap_building_mean = np.mean(boothstrap_building, axis=1) # calculating
mean of building for each of the 10000 bootstraps
    bootstrap_face_mean = np.mean(boothstrap_face, axis=1) # calculating mean of
face for each of the 10000 bootstraps
```

```
    bootstrap_mean_difrence = bootstrap_building_mean - bootstrap_face_mean # cal
culating mean difrence for each of the 10000 bootstraps, this form the mean difre
nce distribution of the population.

    plt.figure() # I wanted to show how it looks
    plt.title('Building, Face Mean Difference Distribution \nFor the Same Partici
pants')
    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difrence, bins=29, density=True)

    given_mean_difrence = np.mean(building)-
np.mean(face) # computing the mean difrence for the origanal pop1, pop2 given to 
us

    p = 2 * (1 - norm.cdf(( ( np.abs(given_mean_difrence) - np.mean(bootstrap_mea
n_difrence)) / (np.std(bootstrap_mean_difrence)) ))) # here we compute P(|bootstr
ap_mean_difrence| > |given_mean_difrence|) by using the CDF of the standart norml
a distribution.


    significant_points = np.where(np.abs(bootstrap_mean_difrence) > np.abs(given_
mean_difrence)) # finding the significant points


    two_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 2 t
ailed p value


    print("Two-tailed p-
value for the null hypothesis that building and face create the same response is 
" + str(two_tailed_p_value) + " for counting bootstrap results over the given dif
ference. \nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and b
uildings and faces create different responses.")


def question2_part_e():

    print("\nAnswer of question 2 part e: \n")

    data2 = sio.loadmat('matlab_HW3_data3') # loading the data given to us
    building = data2["building"]
    face = data2["face"]

    # null hypothesis is they have no difference in their response. If they have
no difference in their response then their difrence in means will be zero. So we
will compute the difference in means distribution asuming they have no difrence i
n their responses. then I will find the probability of obtaining the same difrenc
e in means as the original given data.
    # what we do in this question is same as part a.

    building_face = np.concatenate((building, face), axis = 1) # the neron group
from our null hypotezis

    bootstraps = bootstrap_singel(10000, building_face.transpose()) # creating 10
000  bootstraps from the total population

    bootstrap_building = bootstraps[:,0:20] # picking the first 20 elements as th
e building for each of the 10000 bootstraps
    bootstrap_face = bootstraps[:,20:40] # picking the last 20 elements as the fa
ce for each of the 10000 bootstraps

    bootstrap_building_mean = np.mean(bootstrap_building, axis=1) # calculating m
ean of building experiment for each of the 10000 bootstraps
    bootstrap_face_mean = np.mean(bootstrap_face, axis=1) # calculating mean of f
ace experiment for each of the 10000 bootstraps

    bootstrap_mean_difference = bootstrap_building_mean - bootstrap_face_mean # c
alculating mean difference for each of the 10000 bootstraps, this form the mean d
ifference distribution of the population.


    given_mean_difrence = np.mean(building)-
np.mean(face) # computing the mean difrence for the origanal building, face given
 to us

    p = 2 * (1 - norm.cdf(( ( np.abs(given_mean_difrence) - np.mean(bootstrap_mea
n_difference) ) / (np.std(bootstrap_mean_difference)) ))) # here we compute P(|bo
otstrap_mean_difrence| > |given_mean_difrence|) by using the CDF of the standart
normla distribution.


    significant_points = np.where(np.abs(bootstrap_mean_difference) > np.abs(give
n_mean_difrence)) # finding the significant points

    two_tailed_p_value = np.size(significant_points[0])/10000 # calculatin of 2 t
ailed p value
```

```
    print("Two-tailed p-
value for the null hypothesis that building and face create the same response is 
" + str(two_tailed_p_value) + " for counting bootstrap results over the given dif
ference. \nalso the p-
value is " + str(p) + " for fitting a gausian to the null hypotehseies distiribui
ton. \nThis is significantly lower than 0.05 so this hypothesis is incorect and b
uildings and faces create different responses.")
    print("\nThis means there is a different response for building and face, rega
rdless of our assumption of the participitants.")

    plt.figure() # I wanted to show how it looks
    plt.title('Building, Face Response Mean Difference Distribution \nFor Differe
nt Participants')
    plt.xlabel('E[pop1] - E[pop2]')
    plt.ylabel('Probability')
    plt.yticks([])
    plt.hist(bootstrap_mean_difference, bins=29, density=True)
    plt.show() # this privents the plot from closing themselves at the end


Batu_Arda_Düzgün_21802633_hw3(question) # this is the only line which runs so it 
is very important.
```