## Problem 1

1. To get a bootstrap estimate of bias and standard error of median, I have written a python function boot\_bias\_sde. This function estimate bias and standard error for both for mean and median. Before, describing the function I will discuss about the algorithm that I have used to get the results. I have used the following algorithm.
2. Estimate the parameter (mean / median)
3. Generate bootstrap samples (say ) times and estimate the parameter (mean / median) for each bootstrap sample and assign those estimates in a list
4. From that list of parameter obtained in step (b), find bias and standard error and return results.

This function boot\_bias\_sde has four parameters-

x *= input data for which we want to find bias and standard error*

boottimes = *number of bootstrap sample (Default 10)*

method = *parameter (mean / median) (Default “Mean”) for which we want to estimate bias and standard error*

size = *Bootstrap sample size (Default “None” that means bootstrap sample size will be the size of the input data.*

# -\*- coding: utf-8 -\*-  
"""  
Created on Sun Nov 13 14:00:27 2016  
  
@author: Kanak  
"""  
  
##################################################################  
# Problem 1 Part 1  
##################################################################

At first I have imported two python library “pandas” and “math” and then using pandas function pd.read\_csv, I have import the data file as “Data Frame” and assign the data frame in df.

import pandas as pd  
import math  
  
df = pd.read\_csv("D:/UCF/STA 6106 Statistical Computing/Assignments/Midterm 2/Exam2\_pb1.txt", header=None, names=["a"])

Following “boot\_bias\_sde” is the function for finding bias and standard error of a parameter.

def boot\_bias\_sde(x, boottimes = 10, method = 'Mean', size = None):

Here I have first check where the input data in this function is data frame or not. If the input data is not a data frame, it will try to convert the data into a data frame. However, if there is any error while converting as a data frame, it will print a massage ("x is not possible to convert as Data Frame") and exit from the function immediately. For doing this I have used error handler (try- except) attribute.

**Note:** It is important to mention that the python functions that I have used in this problem, is based on data frame type of data. This is why it is important to check data type. For example, if we pass list type of data in this function it will create error.

if not isinstance(x, pd.DataFrame):  
 try:  
 x = pd.DataFrame(x)  
 except:  
 print ("x is not possible to convert as Data Frame")  
 return

Then, it will check whether bootstrap sample size (size) is None or not. If it is None then it will consider the input data size as the bootstrap sample size.

if size == None:  
 size = x.shape[0]

Now, based on method type (Mean / Median) it will calculate step (a) and (b) and assign step (a), estimate the parameter (mean / median) as theta. To find the bootstrap estimate for each bootstrap sample I have used a loop. The way I have written this code is called one-liners. This loop will continue boottimes (say 9999) times and within each loop it will take a sample from the given data with replacement and then calculate the parameter (mean / median) and assign each parameter (mean / median) in a list called thetab. Finally, it will convert the list of bootstrap parameters (mean / median) as a data frame.

if method.upper() == 'MEAN':  
 theta = x.mean()  
 thetab = pd.DataFrame([(x.sample(n = size, replace= True)).mean()\  
 for \_ in range(boottimes)])  
 if method.upper() == 'MEDIAN':  
 theta = x.median()  
 thetab = pd.DataFrame([(x.sample(n=size, replace= True)).median()\  
 for \_ in range(boottimes)])

After getting bootstrap parameter (mean / median) data frame, I have calculated the bias and standard error by the following formula.

where,

And where,

botse = thetab.std()  
 bias = thetab.mean() – theta

Finally, I have returned a python dictionary containing parameter estimate, bias and standard error. Since, we have used data frame for in all the above steps, to return those results as array type data I have used values and flatten functions.

return {"Parameter": theta.values.flatten(),  
 "Bias": bias.values.flatten(),  
 "Standard Error": botse.values.flatten()}

To have bootstrap bias and standard error of estimate (mean / median), I have called the function boot\_bias\_sde with function parameters df, boottimes = 9999, method = 'median' and size = None. The output of this function is given below.

a = boot\_bias\_sde(df, boottimes = 9999, method = 'median', size = None)  
print (a)

## {'Bias': array([ 8.815]), 'Standard Error': array([ 58.8009]),

## 'Parameter': array([ 331.5])}

**Note:** This function works for any dimension of data. For example if we pass data with two variables, it will calculate bootstrap Bias and standard error of parameter (mean / median) for each of those variables.

1. To get a jackknife estimate of bias and standard error of mean, I have written a python function jackknife. This function estimate bias and standard error for both for mean and median. Before, describing the function I will discuss about the algorithm that I have used to get the results. I have used the following algorithm.
2. Estimate the parameter (mean / median)
3. Generate jackknife samples (say size of input data) times by dropping observation () and estimate the parameter (mean / median) for each jackknife sample and assign those estimates in a list
4. From that list of parameter obtained in step (b), find bias and standard error and return results.

This function jackknife has two parameters-

x *= input data for which we want to find jackknife bias and standard error*

method = *parameter (mean / median) (Default “Mean”) for which we want to estimate bias and standard error*

Following “jackknife” is the function for finding bias and standard error of a parameter.

##################################################################  
# Problem 1 Part 2  
##################################################################

def jackknife(x, method = 'Mean'):

Here I have first check where the input data in this function is data frame or not. If the input data is not a data frame, it will try to convert the data into a data frame. However, if there is any error while converting as a data frame, it will print a massage ("x is not possible to convert as Data Frame") and exit from the function immediately. For doing this I have used error handler (try- except) attribute.

**Note:** It is important to mention that the python functions that I have used in this problem, is based on data frame type of data. This is why it is important to check data type. For example, if we pass list type of data in this function it will create error.

if not isinstance(x, pd.DataFrame):  
 try:  
 x = pd.DataFrame(x)  
 except:  
 print ("x is not possible to convert as Data Frame")  
 return

Now, based on method type (Mean / Median) it will calculate step (a) and (b) and assign step (a), estimate the parameter (mean / median) as theta. To find the bootstrap estimate for each bootstrap sample I have used a loop. The way I have written this code is called one-liners. This loop will continue (size of input sample) times and within each loop it will take a sample from the given data leaving one observation from the input sample and then calculate the parameter (mean / median) and assign each parameter (mean / median) in a list called thetab. Finally, it will convert the list of jackknife parameters (mean / median) as a data frame.

n = x.shape[0]  
 if method.upper() == 'MEAN':  
 theta = x.mean()  
 thetab = pd.DataFrame([(x.drop(i)).mean() for i in range(n)])  
 if method.upper() == 'MEDIAN':  
 theta = x.median()  
 thetab = pd.DataFrame([(x.drop(i)).median() for i in range(n)])

After getting jackknife parameter (mean / median) data frame, I have calculated the bias and standard error by the following formula.

where,

and where,

jkse = ((n-1)/math.sqrt(n))\*thetab.std()  
 bias = (n-1)\*(thetab.mean() - theta)

Finally, I have returned a python dictionary containing jackknife parameter estimate, bias and standard error. Since, we have used data frame for in all the above steps, to return those results as array type data I have used values and flatten functions.

return {"Parameter": theta.values.flatten()),   
 "Bias": bias.values.flatten(),  
 "Standard Error": jkse.values.flatten()}

To have jackknife bias and standard error of estimate (mean / median), I have called the function jackknife with function parameters df, method = "mean". The output of this function is given below.

jk = jackknife(df, method = "mean")  
print(jk)

## {'Bias': array([ 0.]), 'Standard Error': array([ 50.7908]),

## 'Parameter': array([ 437.2097])}

**Note:** This function works for any dimension of data. For example, if we pass data with two variables, it will calculate jackknife Bias and standard error of parameter (mean / median) for each of those variables.

1. To get a bootstrap estimate of bias and standard error of median and bootstrap confidence interval, I have written a python function boot\_ci. The “boot\_ci” function estimate bias and standard error and its percentile and bootstrap confidence interval for both for mean and median. Before, describing the function I will discuss about the algorithm that I have used to get the results. I have used the following algorithm.
2. Estimate the parameter (mean / median)
3. Generate bootstrap samples (say ) times and estimate the parameter (mean / median) for each bootstrap sample and assign those estimates in a list
4. From that list of parameter obtained in step (b), find bias and standard error and return results.
5. From that list of parameter obtained in step (b), find and percentile for percentile confidence interval
6. From that list of parameter obtained in step (b), find -statistic and then find and percentile value of -statistic and calculate bootstrap confidence interval

##################################################################  
# Problem 1 Part 3  
##################################################################

Following “boot\_ci” is the function for finding bias and standard error of a parameter and it’s bootstrap and percentile confidence interval.

This function boot\_ci has five parameters-

x *= input data for which we want to find bias and standard error*

boottimes = *number of bootstrap sample (Default 500)*

method = *parameter (mean / median) (Default “Mean”) for which we want to estimate bias and standard error*

r = *Number Bootstrap sample for estimating standard error for each bootstrap sample (Default “100”)*

alpha= *level of significant (Default* 0.05*)*

def boot\_ci(x, alpha= 0.05, boottimes = 10, r = 100, method = 'Mean’):

Here I have first check where the input data in this function is data frame or not. If the input data is not a data frame, it will try to convert the data into a data frame. However, if there is any error while converting as a data frame, it will print a massage ("x is not possible to convert as Data Frame") and exit from the function immediately. For doing this I have used error handler (try- except) attribute.

**Note:** It is important to mention that the python functions that I have used in this problem, is based on data frame type of data. This is why it is important to check data type. For example, if we pass list type of data in this function it will create error.

if not isinstance(x, pd.DataFrame):  
 try:  
 x = pd.DataFrame(x)  
 except:  
 print ("x is not possible to convert as Data Frame")  
 return

Then, it will consider the input data size as the bootstrap sample size.

size = x.shape[0]

Here, I have written a function boot\_se to find the bootstrap standard error for each bootstrap sample. This is almost similar to problem 1 part 1.

def boot\_se(y, r, method, size):  
 if method.upper() == 'MEAN':  
 thetab = pd.DataFrame([(y.sample(n=size, replace=True)).mean() \  
 for \_ in range(r)])  
 if method.upper() == 'MEDIAN':  
 thetab=pd.DataFrame([(y.sample(n=size, replace=True)).median() \  
 for \_ in range(r)])  
 return thetab.std().values.flatten()

Now, I have created two null list to store standard error and the estimate for each bootstrap sample. Then, based on method type (Mean / Median) it will estimate the parameter (mean / median) and store it to theta. To find the bootstrap estimate for each bootstrap sample I have used a loop. This loop will continue boottimes (say 9999) times and within each loop it will take a sample from the given data with replacement and then calculate standard error using boot\_se and the parameter (mean / median) for that sample and it will append those standard error and parameter estimate to list se and thetab respectively.

se = []  
 thetab = []  
 if method.upper() == 'MEAN':  
 theta = x.mean().values.flatten()  
 for \_ in range(boottimes):  
 y =x.sample(n = size, replace = True)  
 se.append(boot\_se(y, r, method, size))  
 thetab.append(y.mean().values.flatten())   
  
 if method.upper() == 'MEDIAN':  
 theta = x.median().values.flatten()  
 for \_ in range(boottimes):  
 y =x.sample(n = size, replace = True)  
 se.append(boot\_se(y, r, method, size))  
 thetab.append(y.median())

After getting data frame of bootstrap parameters (mean / median), I have calculated -statistic and then converted the list of bootstrap parameter to a data frame. The bias and standard error were calculated by the following formula. **It is important to mention that for all results I have used data frame function and to convert data frame as an array, I have used values and flatten functions.**

where,

And where,

t = pd.DataFrame((thetab - np.mean(thetab))/ se)  
 thetab = pd.DataFrame(thetab)  
 bias = (thetab - theta).mean()

sd = thetab.std().values.flatten()

Again, to find and percentile confidence interval, I have used the data frame of the bootstrap parameters (mean / median) and used quantile function with quantiles (alpha/2, 1-alpha/2).

qinterval = thetab.quantile(q = (alpha/2, 1-alpha/2))

Similarly, to find bootstrap confidence interval, I have used quantile function to find the simulated and percentile of -statistics and using abs() function, I got absolute -statistics. Finally, using following equation, bootstrap confidence interval has been calculated.

and

where and are the and percentile of -statistics of bootstrap parameters (mean / median).

tqt = t.quantile(q = (alpha/2, 1-alpha/2)).abs().values.flatten()  
 bci = theta + (tqt \* [-1, 1] \* sd)

Finally, I have returned a python dictionary containing parameter estimate, bias and standard error and percentile and bootstrap confidence interval.

ci = {"Parameter": theta,  
 "Bootstrap CI": bci,   
 "Percentile Interval": qinterval.values.flatten(),  
 "Bias": bias.values.flatten(),  
 "Standard Error": sd}

To have bootstrap bias and standard error of estimate (mean / median) and it’s percentile and bootstrap confidence interval, I have called the function boot\_bias\_sde with function parameters df, alpha = 0.05, boottimes = 9999, method = 'median'. The output of this function is given below.

ci = boot\_ci(df, alpha = 0.05, boottimes = 5000, method = 'median')  
print(ci)

## {'Bias': array([ 8.4198]), 'Standard Error': array([ 57.71608959]),

## 'Bootstrap CI': array([ 206.51556028, 430.79777328]), 'Percentile

## Interval': array([ 224.5, 454. ]), 'Parameter': array([ 331.5])}

**Note:** This function works for any dimension of data. For example if we pass data with two variables, it will calculate bootstrap Bias and standard error of parameter (mean / median) and it’s Percentile and Bootstrap Confidence interval for each of those variables.

## Problem 2

We have, the Elastic Net problem:

(1)

We can write the equation (1) as-

where, we consider that are known and our objective is to optimize .

Let be the partial residual

So,

The sub-gradient can be obtained by taking the derivative with respect to -

If we consider that is standardized, then

So,

Setting sub-gradient to 0 gives-

when

when

when

So,

## Problem 3

1. To detect outliers or abnormal observations, I have written three python function tstat and baggging and predictoutlier. “tstat” function estimate -statistic for a given value with mean and standard deviation, “baggging” function estimate the threshold value and predictoutlier estimate can be used to detect outlier prediction. I have used the algorithm given in problem.

##################################################################  
# Problem 3  
##################################################################

At first I have imported two python library “pandas” and “numpy” and the data list.

import numpy as np  
import pandas as pd  
x = [28, -44, 29, 30,26, 27,22, 23, 33,16, 24,40, 21,31, 34,-2, 25, 19]

This function tstat has four parameters-

x *= input data*

theta = *mean of sample (Default* None*)*

sigsq = *standard deviation (Default* None*)*

def tstat(x, theta = None, sigsq = None):

In this function, it will first check whether there are any values of mean or standard deviation have been supplied in the function or not. If not, it will calculate the mean and standard deviation from the given sample. Finally, it will return -statistics calculated using following equation.

if theta is None:  
 theta = x.mean()  
 if sigsq is None:  
 sigsq = x.std()  
 return ((x-theta)/sigsq)

Following “baggging” is the function for finding threshold value for a given sample. This function baggging has four parameters-

x *= input data*

boottimes = *number of bootstrap sample (Default 100)*

size = *Bootstrap sample size (Default “None” that means bootstrap sample size will be the size of the input data.*

alpha= *level of significant (Default* 0.05*)*

def baggging(x, size = None, alpha = 0.05, boottimes = 100):

Here, I have first check where the input data in this function is data frame or not. If the input data is not a data frame, it will try to convert the data into a data frame. However, if there is any error while converting as a data frame, it will print a massage ("x is not possible to convert as Data Frame") and exit from the function immediately. For doing this I have used error handler (try- except) attribute.

**Note:** It is important to mention that the python functions that I have used in this problem, is based on data frame type of data. This is why it is important to check data type. For example, if we pass list type of data in this function it will create error.

if not isinstance(x, pd.DataFrame):  
 try:  
 x = pd.DataFrame(x)  
 except:  
 print ("x is not possible to convert as Data Frame")  
 return

Then, it will check whether bootstrap sample size (size) is None or not. If it is None then it will consider the input data size as the bootstrap sample size.

n = x.shape[0]  
 if size == None:  
 size = n

First, we take a bootstrap random sample from input data with replacement. Then, mean and variance were calculated using mean() and var() function for data frame. I have used values and flatten() function to convert data frame to an array. Using that mean and variance, I have calculated the t-statistic using tstat function that I have discussed earlier. For this t-statistic data frame, percentile is calculated and assigned it to hi as an array. This process has been repeated for (boottimes) times. Finally, after converting this hi to a data frame, mean has been calculated and converted the result as an array to find the value of .

hi = []  
 for \_ in range(boottimes):  
 y = x.sample(n = size, replace = True)  
 m = y.mean().values.flatten()  
 sd = y.var().values.flatten()  
 t = tstat(y, theta = m, sigsq = sd)  
 hi.append(t.quantile(q = 1-alpha).values.flatten())  
 h = pd.DataFrame(hi).mean()

Finally, I have returned a python dictionary containing threshold value , mean and variance of the given sample.

return {"h": h.values.flatten(),  
 "Mean": x.mean().values.flatten(),  
 "Variance": x.var().values.flatten()}

To have threshold value , I have called the function baggging with function parameters x, boottimes = 9999. The output of this function is given below.

model = baggging(x, boottimes = 9999)  
print(model)

## {'Variance': array([ 343.59477124]), 'h': array([ 4.01523935]),

## 'Mean': array([ 21.22222222])}

**Note:** This function works for any dimension of data. For example, if we pass data with two variables, it will calculate value for each of those variables.

1. To predict a new observation whether it is outliers or abnormal observation or not, I have written predictoutlier function.

Function predictoutlier has two parameters-

model *= model obtained from* baggging *function*

x = *new observation that we want to predict*

def predictoutlier(model, x):

At first, all parameters have been assigned to variables. Then, using tstat function, t-statistic has been calculated for new observation and assign to variable.

par = model['h']  
 m = model["Mean"]  
 sd = model["Variance"]  
 t = np.array(tstat(x, m, sd))

After getting t-statistic, we have compared it with threshold value that is if t-statistic is greater than 0 and less than , it is not an outlier. Since this function can be used for multivariate data, that is why I have used minimum (np.ndarray.min) and maximum (np.ndarray.max) function for array data and the logic is, if any one of the multivariate t-statistic is greater than any one of the multiple threshold values , it is an outlier. Based on the condition, it will make comment on the observation.

**Note:** This function works for any dimension of data. For example, if we pass data with two variables, it will calculate value for each of those variables.

if np.ndarray.min(t)< 0 or np.ndarray.max(t) > np.ndarray.min(par):  
 out = "Outlier"  
 com = ("The observation {} with t-value {} is an OUTLIER "+  
 *"* because *given value does not fall between 0 “*+

*“and {}"*).format(x, t, par)

print(com)  
 else:  
 out = "Not Outlier"  
 com = ("The observation {} with t-value {} is NOT OUTLIER" +  
 *" because given value falls between 0 “* +

*“and {}"*).format(x, t, par)

print(com)

Finally, I have returned a python dictionary containing threshold value , mean and variance, t-statistic for new observation, decision whether new observation is an outlier or not and comment about the prediction.

return {"h": par,  
 "Mean": m,  
 "Variance": sd,  
 "t": t,   
 "Decision": out,   
 "Comment": com}

To predict new observation 38, I have called the function predictoutlier with function parameters model that was obtained by calling the function baggging and the new observation 38. The output of this function is given below.

pred = predictoutlier(model, 38)  
print(pred)

## {'h': array([ 4.01523935]), 'Comment': 'The observation 38 with

## t-value [ 0.81926109] is NOT OUTLIER because given value falls

## between 0 and [ 4.01523935]', 'Mean': array([ 21.22222222]),

## 'Variance': array([ 343.59477124]), 'Decision': 'Not Outlier',

## 't': array([ 0.81926109])}

It is found that new observation 38 is not an outlier.