STATS 202: Data Mining and Analysis
Homework #2

Adam Kainikara

July 17, 2023

Problem 1 (5 points)
Chapter 4, Exercise 1 (p. 189).
4.2 $p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$
4.3 $\frac{p(X)}{1-p(X)} = e^{\beta_0+\beta_1 X}$
Starting with 4.3
$p(X) = (1-p(X))e^{\beta_0+\beta_1 X}$
$p(X) = e^{\beta_0+\beta_1 X} - e^{\beta_0+\beta_1 X}p(X)$
$p(X) + e^{\beta_0+\beta_1 X}p(X) = e^{\beta_0+\beta_1 X}$
$p(X)(1+e^{\beta_0+\beta_1 X}) = e^{\beta_0+\beta_1 X}$
$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$
This is 4.2
Therefore 4.2 is equivalent to 4.3
Problem 2 (5 points)
Chapter 4, Exercise 4 (p. 189).
a) Because we are using 10% of observations in the range of X. This is the same as [100x +5)%. To find the average, we will have to integrate it.
$\int_{0.05}^{0.95} 10x dx + \int_{0}^{0.05} 10x + 5\, dx + \int_{0.95}^{1} 105 - 100x\, dx = 9.75$
b) X1 and X2 are independent. Under the same assumptions the fraction of available observations would be $9.75\%^2 = \sim 0.95\%$
c) This is similar to (b) just with 100 times. $9.75\%^{100} = \sim 0\%$
d) As the p increases, the fraction of amiable observations tends to 0.
e) p=100 $l = 0.1^{0.01}$
Problem 3 (5 points)
Chapter 4, Exercise 6 (p. 191).
a) $\hat{\beta}_0 = -6\hat{\beta}_1 = 0.05\hat{\beta}_2 = 1$ are the coefficients for this question. These are coefficients for the constant, number of hours studied and undergrad GPA. Plugging in the values for this particular student gives $-6 + (0.05 \times 40) + (1 \times 3.5) = -0.5$. This value can be plugged into the logistic formula to calculate the probability.
$\hat{p}(x) = \frac{e^x}{1+e^x}$
$\hat{p}(x) = \frac{e^{-0.5}}{1+e^{-0.5}} = 0.378$

A student who studied for 40 hours and has an undergrad GPA of 3.5 has about a 37.8% chance of getting an A in the class.

b) How long does the student need to study for a 50% chance of an A.

Want $\hat{p}(x) = 0.5$

$0.5 = \frac{e^x}{1+e^x}$

$\frac{1}{2e^x} = \frac{1}{1+e^x}$

$2e^x = 1 + e^x$

$x = 0$

$0 = -6 + 0.05X_1 + 1X$

Because its the same student and only the hours studied is changing X_2(the GPA) is still the same

$0 = -6 + 0.05X_1 + 1(3.5)$

$X_1 = 50$

If this student wants a 50% of an A then they should study for 50 hours.

Problem 4 (5 points)

Chapter 4, Exercise 8 (p. 191).

Using KNN with K = 1, the training error rate can be calculated to be 0%. This means that the test error rate has to be 36% in order for the average to be 18%. Because the logistic regression fit had a test error rate of 20%, we should use the logistic fit because it

has a lower test error rate.

Problem 5 (5 points)

Chapter 4, Exercise 13 parts a-h (p. 193)

a) See bar graph

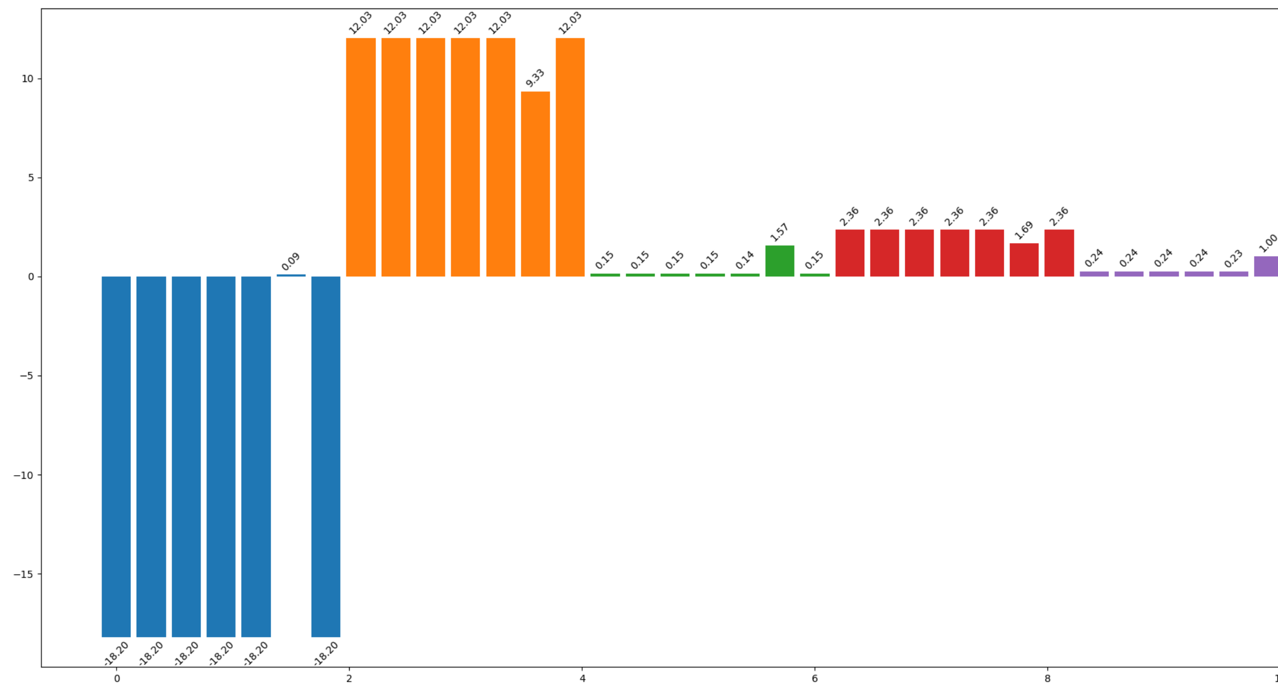

Figure 1: Min, Max, Mean Stand Dev and Median in the Colors Blue, Orange, Green, Red and Purple Respectively

The bar graph shows the min, max, mean stand dev and median in the colors blue, orange, green, red and purple respectively. These values are for Lag 1, 2, 3, 4, 5, Volume and Today. The bar graph shows similar results for each parameter. Only volume seems to change. The numerical summary for this data is below.

min: [-18.195 -18.195 -18.195 -18.195 -18.195 0.087465 -18.195]

max: [12.026 12.026 12.026 12.026 12.026 9.328214 12.026]

mean: [0.15058494 0.15107897 0.14720478 0.14581818 0.13989256 1.57461763 0.14989899]

stand dev: [2.35593008 2.35617168 2.35941794 2.35919491 2.36020027 1.68586184 2.35584499]

median: [0.241 0.241 0.241 0.238 0.234 1.00268 0.241]

b) The coefficients for the logistic regression are below.

[[5.37367327e-02 5.34098644e-03 -1.50782005e-02 5.70986281e-02 9.18621394e-02 7.21885742e+00]]

The logistic regression was performed with direction as the response and the five lag variables plus volume as predictors.

Used stats models API to obtain p values. Lag 2 had a p value of 0.0296 which is lower than 0.05 so the null hypothesis can be rejected. Lag 2 does not influence direction.

c) I first did logistic regression on the whole data set. Then I computed the confusion matrix. The confusion matrix resulted the following:

[[482 2]

[0 605]]

The model was very successful in predicting whether the data would go up or down. The model correctly predicted down every single time. When predicting up, there were only two instances the model predicted wrong of the 607 times. Of all 1089 data points, just two where predicted wrong.

d) Training data is 1990 to 2008 which is until row 986.

I fitted a logistic regression using the training data. I then computed the confusion matrix for the left out data (test data) which was the data from 2009 and 2010. The confusion matrix resulted the following:

Confusion Matrix of the test data with Lag2 as the only predictor

[[9 34]

[5 56]]

Of the 104 data points from 2009 and 2010, 39 (37.5%) where incorrect. 37.5% is the test error rate. In this time period there were 61 ups and 43 downs. Of the 61 ups, 5 were incorrect (8.2%). Of the 43 downs, 34 where incorrect (79.1%).

e, f, g, h, i) Doing d again but with LDA, QDA, KNN.

Confusion Matrix of the training data with Lag2 as the only predictor but instead used LDA

[[22 419]

[20 524]]

Confusion Matrix of the test data with Lag2 as the only predictor but instead used LDA

[[9 34]

[5 56]]

Confusion Matrix of the training data with Lag2 as the only predictor but instead used QDA

[[0 441]

[0 544]]

Confusion Matrix of the test data with Lag2 as the only predictor but instead used QDA

[[0 43]

[0 61]]

Confusion Matrix of the training data with Lag2 as the only predictor but instead used NAIVE BAYES

[[0 441]

[0 544]]

Confusion Matrix of the test data with Lag2 as the only predictor but instead used NAIVE BAYES

[[0 43]

[0 61]]

Confusion Matrix of the training data with Lag2 as the only predictor but instead used KNN

[[298 143]

[106 438]]

Confusion Matrix of the test data with Lag2 as the only predictor but instead used KNN

[[16 27]

[19 42]]

Error Rates (test data)	Lag 2 Logi	LDA	QDA	N
Test Error Rate	37.5%	37.5%	41.3%	
Incorrectly Predicted Down	79.1% (34/43 wrong)	79.1% (34/43 wrong)	100% (43/43 wrong)	100
Incorrectly Predicted Up	8.2 % (5/61 wrong)	8.2 % (5/61 wrong)	0% (0/61 wrong)	0%

Lag 2 Logi only had the lowest test error rate. KNN was the best at predicting down. QDA and NAIVE BAYES was best at picking up.

j) I did some transformations. These where using Lag 2 and 3 and changing KNN to K = 7.

Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors

[[23 418]

[23 521]]

Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor

[[8 35]

[4 57]]

Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors but instead used LDA

[[22 419]

[22 522]]

Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used LDA

[[8 35]

[4 57]]

Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictors but instead used QDA

[[12 429]

[13 531]]

Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used QDA

[[4 39]

[2 59]]

Confusion Matrix of the training data with Lag2 as the only predictor but instead used NAIVE BAYES

[[0 441]

[0 544]]

Confusion Matrix of the test data with Lag2 as the only predictor but instead used NAIVE BAYES

[[0 43]

[0 61]]

Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictor but instead used KNN

[[254 187]

[120 424]]

Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor but instead used KNN [

[11 32]

[17 44]]

Error Rates (test data)	Lag 2 and 3 Logi	LDA	QDA	N
Test Error Rate	37.5	37.5	39.4	
Incorrectly Predicted Down	81.4% (35/43 wrong)	81.4% (35/43 wrong)	90.7% (39/43 wrong)	100
Incorrectly Predicted Up	6.3 % (4/61 wrong)	6.3 % (4/61 wrong)	3.3% (2/61 wrong)	0

Problem 6 (5 points)

Chapter 5, Exercise 2 (p. 219).

a) Let n be the number of observations. The probability that the jth observation is in the bootstrap is $\frac{1}{n}$so the probability the jth observation is not in the probability is $1 - \frac{1}{n}$.

b) Each bootstrap is independent. So the probability that the jth observation is not in the second is the same: $1 - \frac{1}{n}$.

c) Bootstrapping has sample with replacement and is independent. The probability that the jth observation is not in a observation is $1 - \frac{1}{n}$. The probability that the jth observation is not in the bootstrap sample is the product of this. So it becomes $(1 - \frac{1}{n}) \times (1 - \frac{1}{n}) \times \ldots = (1 - \frac{1}{n})^n$

d) n =5, finding if the jth observation is in the bootstrap.

P(observation in the bootstrap) = 1 - P(observation is not in the bootstrap)

$1 - (1 - \frac{1}{5})^5 = 0.67$

e) Same as above but now n = 100

$1 - (1 - \frac{1}{100})^{100} = 0.63$

f) Same as above but now n = 10000

$1 - (1 - \frac{1}{10000})^{10000} = 0.63$

g) Note: For the graph I did n = 20,000 because when I did n = 100,000 my computer crashed and above 30,000 it was taking a very long time for the graph to load.

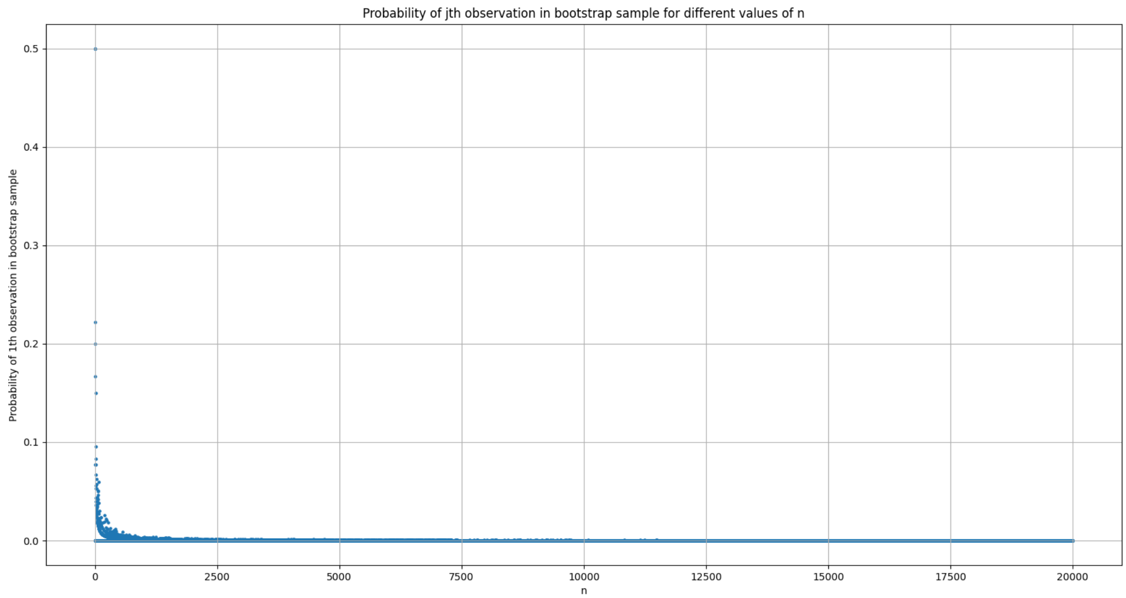

Figure 2: Probability of Jth Observation

The graph asymptotes around 0.63

h) As $n->\infty, p=0.632$

Problem 7 (5 points)

Chapter 5, Exercise 5 (p. 220).

a) The coefficients for income and balance fit using logistic regression to predict default are:

[[5.64710797e-03,

2.08091984e-05]]

b) 1) Did split into test and validation set. The data was split with 5000 observations in each set.

2) Fitted a multi logistic model. Coefficients are:

[[0.00041108

-0.00012325]]

3) Classified using posterior probability

4) Computed the validation set error. The confusion matrix was:

[[4996 1]

[3 0]]

The error for the validation set is 0.08%

c) 1) Did split into test and validation set. The data was split with 7500 observations in the training set and 2500 observations in the validation set.

2) Fitted a multi logistic model.Coefficients are:

[[5.79458398e-03

2.30839912e-05]]

3) Classified using posterior probability

4) Computed the validation set error. The confusion matrix was:

[[7275 107]

[112 6]]

The error for the validation set is 2.2%

5) Did split into test and validation set. The data was split with 2500 observations in the training set and 7500 observations in the validation set.

6) Fitted a multi logistic model. Coefficients are:

[[0.00043135

-0.00012144]]

7) Classified using posterior probability

8) Computed the validation set error. The confusion matrix was:

[[2499 0]

[1 0]]

The error for the validation set is 0.04%

9) Did split into test and validation set. The data was split with 6000 observations in the training set and 4000 observations in the validation set.

10) Fitted a multi logistic model. Coefficients are:

[[5.95395704e-03

3.26622332e-05]]

11) Classified using posterior probability

12) Computed the validation set error. The confusion matrix was:

[[3872 65]

[61 2]]

The error for the validation set is 3.2%

d) I used a list comprehension to switch the "yes" and "no" to 1 and 0. Then added it to the array. Then computed the logistic regression and made a confusion matrix of the validation set. The results are below:

[[4996 1]

[3 0]]

The error for the validation set is 0.08%

Adding a dummy variable, student, did not result in an increase or decrease in the error rate.

Problem 8 (5 points)

Chapter 5, Exercise 6 (p. 221).

a) The standard error for the coefficients balance, income and intercept are as follows.

$Name$	$Standard\,Error\,for\,the\,coefficients$
$balance$	0
$income$	$4.99e-06$
$intercept$	0.435

b) See code

c,d) The standard error was 3.804649256670676.

Problem 9 (5 points)

Chapter 5, Exercise 8 (p. 222).

a) In this problem n = 100 and p = 2. The model in this problem is $Y = X - 2X^2 + constant$

b) See scatter plot. The scatter plot has a upside down parabolic shape. Most of the points occur at the top of the curve.

c, d ,e)

I first fitted these 4 models using linear algebra and coding it.

i	$Y = \beta_0 + \beta_1 X + \epsilon$
ii	$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$
iii	$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon$
iv	$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$

The seed was set to 1:

$rng = np.random.default_rng(1)$

The resulting coefficients for the models are as follows:

	β_0	β_1	β_2	β_3	β_4
i	-1.46496301	1.94936857			
ii	-0.07275529	0.96627276	-2.00470902		
iii	-0.05719669	1.1145842	-2.04709357	-0.06430033	
iv	0.10084766	0.90499786	-2.50592308	0.03376837	0.10421699

I then used LOOCV to help fit the models. The coefficients for each of the models that produced the lowest MSE are as follows:

	β_0	β_1	β_2	β_3	β_4	$Error$
i	-1.46518099	1.94913361				5.9226536056081205
ii	-0.0726598	0.96615516	-2.00472843			0.9834354527808048
iii	-0.05737607	1.11461298	-2.04701671	-0.06429704		0.9718403920130584
iv	0.10079418	0.90506572	-2.50587159	0.03374964	0.10420779	0.9201948751940691

Changed the seed to 100.

$rng = np.random.default_rng(100)$

The resulting coefficients for the models are as follows:

	β_0	β_1	β_2	β_3	β_4
i	-1.69029473	0.50898235			
ii	0.13096828	0.77172222	-1.94030595		
iii	0.15961224	0.46259444	0.46259444	0.13689711	
iv	0.07938098	0.41879764	-1.75905017	0.16392833	-0.06110937

I then used LOOCV to help fit the models. The coefficients for each of the models that produced the lowest MSE are as follows:

	β_0	β_1	β_2	β_3	β_4	$Error$
i	-1.69066004	0.50920579				5.790791169159457
ii	0.13115837	0.7717761	-1.94038458			1.0288896723158567
iii	0.15971326	0.46265244	-1.98187348	0.13688155		0.994999233705984
iv	0.07952053	0.41882023	-1.75921296	0.16391697	-0.0610765	0.9864312854873168

In both seeds model $iv : Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4 + \epsilon$ had the lowest LOOCV. I had mostly expected it as I thought the error would decrease but *iv* would over fit because we knew what the true function was. I had expected that *ii* would have the lower error because it has the same polynomial type as the true function but *iv* does have a lower error it just over fits the data.

f) The coefficients for $\beta_0, \beta_1, \beta_2$ all have p values less than 0.05. This means that these coefficients significantly help predict y. This is what should happen because the original function is a quadratic.

Problem 10 (5 points)

Chapter 5, Exercise 9 (p. 223).

a) The population mean: $\hat{\mu} = 22.53$

b) The standard error of the population mean 0.408. On average the mean of the population will be off from the population mean by 0.408.

c) Using bootstrap I got a standard error of 0.4018559

d) Con Int: [21.752012851895014, 23.31359979632634

e) $\mu_{\hat{med}} = 21.2$

f) Standard Error of Median 0.36652890745478695

g,h) Tenth Percentile (μ̂0.1) of medv: 12.75 with an error of 0.49. This shows how much the 10 percentile median would be off by on average.

```
 _                 ____      _        _                              _      
| |____      ____|___ \ ___| |_ __ _| |_ ___        ___ ___   __| | ___ 
| '_ \ \ /\ / / __) / __| __/ _` | __/ __|      / __/ _ \ / _` |/ _ \
| | | \ V  V / / __/\__ \ || (_| | |_\__ \     | (_| (_) | (_| |  __/
|_| |_|\_/\_/ |_____|___/\__\__,_|\__|___/_____\___\___/ \__,_|\___|
                                         |_____|                     
```

ch4p13.py

```
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats

import sys
import matplotlib

import matplotlib.pyplot as plt
import numpy
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import accuracy_score #works
from matplotlib.ticker import FormatStrFormatter
import statsmodels.api as sm

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

#Adam Kainikara
#This code is for
#CHAPTER 4 QUESTION 13
#THIS IS FOR PROBLEM 5
#OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202


def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(1,2,3,4,5,6,7), delimiter=',')
    direction_v = loadtxt(fname, skiprows=1, usecols=(8), delimiter=',', dtype=str)

    return data_a, direction_v

def summaries0(x_a):
    print("Numerical summaries")
    for i in range(x_a.shape[1]):
        column = x_a[:, i]
        print(f'Column {i + 1}:')
        print(f'   - Min: {min(column)}')
        print(f'   - Max: {max(column)}')
        print(f'   - Mean: {mean(column)}')
        print(f'   - Median: {median(column)}')
        print(f'   - Stand Dev: {std(column)}')
    print(x_a.shape[1])

    summary = empty((8,5))
    storage_l = []

    for i in range(x_a.shape[1]):
        column = x_a[:, i]
        summary = [min(column), max(column), mean(column), median(column), std(column)]
        print(summary)
        storage_l.append(summary)
    return np.array(storage_l)

def calc_summary(x_a) -> dict[str,ndarray]:
    data_d = {
        "min": x_a.min(axis = 0),
        "max": x_a.max(axis = 0),
        "mean": x_a.mean(axis = 0),
        "stand dev": x_a.std(axis = 0),
        "median": median(x_a, axis = 0)
    }
    return data_d

def print_summary(data_d: dict[str,ndarray]):
    for k,v in data_d.items():
        print(f'{k}:', v)

def find_p_values(x_a, direction_v):
    #y_v = direction_v
    #log_reg = sm.Logit(y_v,ee x_a).fieet()

    # Extract the p-values
    #print(log_reg.summary())
    pass

def plot_summary(data_d):
    data_d = {'min': data_d['min'], 'max': data_d['max'], 'mean': data_d['mean'], 'stan dev': data_d['stand dev'], 'median': data_d['median']
    }
    nvar = len(data_d["min"])
    print(data_d)
    w = 0.25
    stride = w + 0.05
    initial_change = 0
    labels = ("Min", "Max", "Mean", "Stan Dev", "Median")
    x_v = arange(nvar) * stride + 0
    print(x_v)

    fig, ax = plt.subplots()

    for atrribute, measurement in data_d.items():
        #shift = w * initial_change
        rectangle = ax.bar(x_v, measurement, width=w, label = atrribute)
        x_v += nvar * stride
        ax.bar_label(rectangle, fmt = '%.02f', rotation = 45, padding=2)
        #i  nitial_change += 0.25
        #ax.set_xticks(x_v , labels)

```

```
    plt.show()

def summaries(x_a):
    print("Numerical summaries")
    #min_x_a = x_a.min(axis = 0)
    print(f' Min:{x_a.min(axis = 0)}, Max{x_a.max(axis = 0)}, Mean{x_a.mean(axis = 0)}, Stand Dev {x_a.std(axis = 0)}, Median{median(x_a, axis = 0)}')

def logi_reg(x_a, direction_v):
    predictors_a = x_a[:,1:7]
    #print(predictors_a)
    #print(x_a.shape)
    response_v = direction_v

    #response_v = x_a[:,8]
    #print(response_v)
    logreg = LogisticRegression()
    logreg.fit(predictors_a, response_v)
    coefficients = logreg.coef_

    print(coefficients)

def make_prediction(x_a, direction_v):
    clf = LogisticRegression()
    clf.fit(x_a, direction_v)
    ypredict_v = clf.predict(x_a)
    return ypredict_v


def compute_confusion_mat(ypredict_v, direction_v):
    truey_v = direction_v
    confu_mat = confusion_matrix(truey_v, ypredict_v)
    print(confu_mat)
    return confu_mat

def lda_prediction(x_v, y_v):
    #This only served to help me write the code, I did it in main otherwise
    clf = lda()
    clf.fit(x_v, y_v)
    ypredict_ldatrain_v = clf.predict(x_v)
    ypredict_ldatest_v = clf.predict(x_v)
    #Rembr to do the confusion matrix after
    return ypredict_ldatrain_v, ypredict_ldatest_v
    #What I ended up doing
    clf = lda()
    clf.fit(xtrain_v, ytrain_v)

    ypredict_ldatrain_v = clf.predict(xtrain_v)
    print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used LDA")
    compute_confusion_mat(ypredict_ldatrain_v, ytrain_v)

    ypredict_ldatest_v = clf.predict(xtest_v)
    print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used LDA")
    compute_confusion_mat(ypredict_ldatest_v, ytest_v)

def qda_prediction(x_v, y_v):
    #This only served to help me write the code, I did it in main otherwise
    clf = qda()
    clf.fit(x_v, y_v)
    ypredict_ldatrain_v = clf.predict(x_v)
    ypredict_ldatest_v = clf.predict(x_v)
    #Rembr to do the confusion matrix after
    return ypredict_ldatrain_v, ypredict_ldatest_v

    #What I ended up doing
    clf = qda()
    clf.fit(xtrain_v, ytrain_v)
    ypredict_qdatrain_v = clf.predict(xtrain_v)
    print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used QDA")
    compute_confusion_mat(ypredict_qdatrain_v, ytrain_v)

    ypredict_qdatest_v = clf.predict(xtest_v)
    print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used QDA")
    compute_confusion_mat(ypredict_qdatest_v, ytest_v)
def naiv_prediction(x_v, y_v):
    #This onl served to elp me write the code, I did it in main otherwise
    clf = GaussianNB()
    clf.fit(x_v, y_v)
    ypredict_nbtrain_v = clf.predict(x_v)
    ypredict_nbtrest_v = clf.predict(x_v)
    return ypredict_nbtrain_v, ypredict_nbtrest_v
def knn_prediction(x_v, y_v):
    neigh = KNeighborsClassifier(n_neighbors=3)
    neigh.fit(x_v, y_v)
    ypredict_knntrain_v = neigh.predict(x_v)
    ypredict_knntest_v = neigh.predict(x_v)
    return ypredict_knntrain_v, ypredict_knntest_v

def main():
    x_a, y_v = data_loader("Weekly.csv")

    xtrain_v = x_a[:,1:2][:985]
    xtest_v = x_a[:,1:2][985:]

    ytrain_v = y_v[:985]
    ytest_v = y_v[985:]


    '''
    Above is the test and training data for x and y for the remainder of this problem. We will first train then do the test data.
    Then do the confusion matrix
    '''

    '''
    This first part (below) is fitting the training data and then predicitng the y value based on the training data. I
    t then computes the confusion matrix based on the training data
    '''
```

```
clf = LogisticRegression()
clf.fit(xtrain_v, ytrain_v)


ytrain_pred_v = clf.predict(xtrain_v)
print("Confusion Matrix of the training data with Lag2 as the only predictor")
compute_confusion_mat(ytrain_pred_v, ytrain_v)


'''
This second part (below) is getting the predicited y value and computing
the confusion matrix based on the fit found earlier and the test data.
'''

ytest_pred_v = clf.predict(xtest_v)
print("Confusion Matrix of the test data with Lag2 as the only predictor")
compute_confusion_mat(ytest_pred_v, ytest_v)

"""
============================================================================================================================================
NOW DOING QDA
"""
clf = lda()
clf.fit(xtrain_v, ytrain_v)
ypredict_ldatrain_v = clf.predict(xtrain_v)
print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used LDA")
compute_confusion_mat(ypredict_ldatrain_v, ytrain_v)

ypredict_ldatest_v = clf.predict(xtest_v)
print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used LDA")
compute_confusion_mat(ypredict_ldatest_v, ytest_v)

"""
============================================================================================================================================
NOW DOING QDA
"""
clf = qda()
clf.fit(xtrain_v, ytrain_v)
ypredict_qdatrain_v = clf.predict(xtrain_v)
print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used QDA")
compute_confusion_mat(ypredict_qdatrain_v, ytrain_v)

ypredict_qdatest_v = clf.predict(xtest_v)
print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used QDA")
compute_confusion_mat(ypredict_qdatest_v, ytest_v)
"""
============================================================================================================================================
NOW DOING NAIVE BAEES
"""
clf = GaussianNB()
clf.fit(xtrain_v, ytrain_v)
ypredict_nbtrain_v = clf.predict(xtrain_v)
print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used NAIVE BAYES")
compute_confusion_mat(ypredict_nbtrain_v, ytrain_v)

ypredict_nbtest_v = clf.predict(xtest_v)
print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used NAIVE BAYES")
compute_confusion_mat(ypredict_nbtest_v, ytest_v)

"""
============================================================================================================================================
NOW DOING KNN
"""
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(xtrain_v, ytrain_v)
ypredict_knntrain_v = neigh.predict(xtrain_v)
print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used KNN")
compute_confusion_mat(ypredict_knntrain_v, ytrain_v)

ypredict_knntest_v = neigh.predict(xtest_v)
print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used KNN")
compute_confusion_mat(ypredict_knntest_v, ytest_v)

"""
============================================================================================================================================
NOW DOING LDA WITH A TWIST
"""
xtrain2_v = x_a[:,1:3][:985]
xtest2_v = x_a[:,1:3][985:]
ytrain_v = y_v[:985]
ytest_v = y_v[985:]
clf = lda()
clf.fit(xtrain2_v, ytrain_v)
ypredict_ldatrain2_v = clf.predict(xtrain2_v)
print("Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors but instead used LDA")
compute_confusion_mat(ypredict_ldatrain2_v, ytrain_v)

ypredict_ldatest2_v = clf.predict(xtest2_v)
print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used LDA")
compute_confusion_mat(ypredict_ldatest2_v, ytest_v)
"""
============================================================================================================================================
NOW DOING QDA WITH A TWIST
"""
clf = qda()
clf.fit(xtrain2_v, ytrain_v)
ypredict_qdatrain2_v = clf.predict(xtrain2_v)
print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictors but instead used QDA")
compute_confusion_mat(ypredict_qdatrain2_v, ytrain_v)

ypredict_qdatest2_v = clf.predict(xtest2_v)
print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used QDA")
compute_confusion_mat(ypredict_qdatest2_v, ytest_v)
"""
============================================================================================================================================
```

```
    NOW DOING NAIVE BAEES WITH A TWIST
    """
    clf = GaussianNB()
    clf.fit(xtrain2_v, ytrain_v)
    ypredict_nbtrain2_v = clf.predict(xtrain2_v)
    print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictor but instead used NAIVE BAYES")
    compute_confusion_mat(ypredict_nbtrain2_v, ytrain_v)

    ypredict_nbtest2_v = clf.predict(xtest2_v)
    print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor but instead used NAIVE BAYES")
    compute_confusion_mat(ypredict_nbtest2_v, ytest_v)


    """
    ===========================================================================================================================================
    NOW DOING KNN WITH A TWIST
    """
    neigh = KNeighborsClassifier(n_neighbors=7)
    neigh.fit(xtrain2_v, ytrain_v)
    ypredict_knntrain2_v = neigh.predict(xtrain2_v)
    print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictor but instead used KNN")
    compute_confusion_mat(ypredict_knntrain2_v, ytrain_v)

    ypredict_knntest2_v = neigh.predict(xtest2_v)
    print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor but instead used KNN")
    compute_confusion_mat(ypredict_knntest2_v, ytest_v)

    clf = LogisticRegression()
    clf.fit(xtrain2_v, ytrain_v)


    ytrain_pred2_v = clf.predict(xtrain2_v)
    print("Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors")
    compute_confusion_mat(ytrain_pred2_v, ytrain_v)


    ytest_pred2_v = clf.predict(xtest2_v)
    print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor")
    compute_confusion_mat(ytest_pred2_v, ytest_v)




    #print(xtest_a)
    #print(ytrain_v)

    #ytest_v = y_v[:ntest]
    #xtrain_v = x_a[ntest:]

    #confusion_pract()
    #print(x_a)
    #logi_reg(x_a, y_v)
    #find_p_values(x_a, direction_v)
    #summaries(x_a)
    summary = summaries(x_a)
    #calc_summary(x_a)
    data_dict = calc_summary(x_a)
    print_summary(data_dict)
    plot_summary(data_dict)


    clf = LogisticRegression()
    clf.fit(xtrain_v, ytrain_v)
    ytrain_pred_v = clf.predict(xtrain_v)
    compute_confusion_mat(ytrain_pred_v, ytrain_v)
    ytest_pred_v = clf.predict(xtest_v)
    compute_confusion_mat(ytest_pred_v, ytest_v)
    print('Done-----')

    make_prediction(x_a, y_v)
    ypredict_v = make_prediction(x_a, y_v)
    print(ypredict_v)
    compute_confusion_mat(ypredict_v, y_v)
    #print(type(summary))
    #print(summary.shape)
    #plot_summaries(summary)



#Some Links I used
#https://scikit-learn.org/stable/auto_examples/classification/plot_lda_qda.html#sphx-glr-auto-examples-classification-plot-lda-qda-py
#https://stackoverflow.com/questions/46775155/importerror-no-module-named-sklearn-lda

if __name__ == '__main__':
        main()
```

ch5p2.py

```python
from numpy import *
import matplotlib.pyplot as plt

j = 1
num_samples = 20000

n_values = arange(1, num_samples + 1)
probabilities = []

for n in n_values:
    sample = random.choice(range(n), size=n, replace=True)
    probability = mean(sample == j)
    probabilities.append(probability)

# Creating a scatter plot
plt.scatter(n_values, probabilities, s=5)
plt.xlabel('n')
plt.ylabel(f'Probability of {j}th observation in bootstrap sample')
plt.title('Probability of jth observation in bootstrap sample for different values of n')
plt.grid(True)
plt.show()
```

ch5p5.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats

import sys
import matplotlib

import matplotlib.pyplot as plt
import numpy
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import accuracy_score #works
from matplotlib.ticker import FormatStrFormatter
import statsmodels.api as sm

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score


#Adam Kainikara
#This code is for
#CHAPTER 5 QUESTION 5
#THIS IS PROBLEM 7
#OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202

def data_loader(fname):
    num_data_a = loadtxt(fname,skiprows=1, usecols=(2,3), delimiter=',')
    defa_student_a = loadtxt(fname, skiprows=1, usecols=(0,1), delimiter=',', dtype=str)

    return num_data_a, defa_student_a

def logi_reg(x_a, y_a):
    predictors_a = x_a
    response_v = y_a[:,0]
    #print(predictors_a, response_v)
    logreg = LogisticRegression()
    logreg.fit(predictors_a, response_v)
    coefficients = logreg.coef_
    print(coefficients)
def new_logi_reg(x_a, y_a):
    pass

def compute_confusion_mat(ypredict_v, direction_v):
    truey_v = direction_v
    confu_mat = confusion_matrix(truey_v, ypredict_v)
    print(confu_mat)
    return confu_mat

def main():

    x_a, y_a = data_loader("Default.csv")


    ydefault_v = y_a[:,0]
    print(y_a[:,1])
    #ysudent = [1 if x == "yes" else 0 for x in x_a[:,1]]
    ystudent = [1 if x == "Yes" else 0 for x in y_a[:,1]]
    #print(ystudent)

    #print(x_a)
    student_a = array(ystudent)
    #print(student_a.shape)
    xall_a = array([x_a[:,0], x_a[:,1],student_a])
    realx_a = transpose(xall_a)
    print(realx_a.shape)
    print(realx_a)

    xalltrain_a = realx_a[:5000]
    xallvalid_a = realx_a[5000:]
    print(xalltrain_a, xallvalid_a)
    ytrain_v = ydefault_v[:5000]
    yvalid_v = ydefault_v[5000:]
    clf = LogisticRegression()
```

```
    clf.fit(xalltrain_a, ytrain_v)
    ytrain_pred_v = clf.predict(xalltrain_a)
    yvalid_pred_v = clf.predict(xallvalid_a)
    coefficients = clf.coef_
    print(coefficients)
    compute_confusion_mat(ytrain_pred_v, yvalid_pred_v)
    raise SystemExit

    xall_a = ([x_a],[ysudent])
    print(xall_a)
    #logi_reg(x_a,y_a)


    xtrain_a = x_a[:5000]
    xvalid_a = x_a[5000:]
    ytrain_v = ydefault_v[:5000]
    yvalid_v = ydefault_v[5000:]



    clf = LogisticRegression()
    clf.fit(xtrain_a, ytrain_v)
    ytrain_pred_v = clf.predict(xtrain_a)
    yvalid_pred_v = clf.predict(xvalid_a)
    coefficients = clf.coef_
    print(coefficients)
    #compute_confusion_mat(ytrain_pred_v, yvalid_pred_v)


    clf = GaussianNB()
    clf.fit(xtrain_a, ytrain_v)
    posterior_probs = clf.predict_proba(xvalid_a)
    predictions = (posterior_probs > 0.5)
    print(predictions)
    accuracy = accuracy_score(yvalid_v, predictions)
    print("Accuracy:", accuracy)
    #print(xtrain_a, ytrain_v)
    raise SystemExit

    ytrain_v = y_v[:985]
    #ytest_v = y_v[985:]




if __name__ == '__main__':
    main()
```

ch5p6.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats

import sys
import matplotlib

import matplotlib.pyplot as plt
import numpy
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
#from sklearn.metrics import accuracy_score #works
from matplotlib.ticker import FormatStrFormatter
import statsmodels.api as sm

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

import statsmodels.api as sm
rng = np.random.default_rng()
from scipy.stats import norm
#Adam Kainikara
#This code is for
#CHAPTER 5 QUESTION 6
#THIS IS PROBLEM 8
#OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202

def data_loader(fname):
    num_data_a = loadtxt(fname,skiprows=1, usecols=(2,3), delimiter=',')
    defa_a = loadtxt(fname, skiprows=1, usecols=(0,1), delimiter=',', dtype=str)
    ydefault = [1 if x == "Yes" else 0 for x in defa_a[:,0]]
    ystudent = [1 if x == "Yes" else 0 for x in defa_a[:,1]]

    default_a = transpose(array((ydefault, ystudent)))
    #print(default_a)
    return num_data_a, default_a

def use_sm(x_a, y_a):

    b = ones((10000,1))
    xareal_a = hstack((x_a,b))
    print(y_a.dtype)

    print(xareal_a)
    logit_model = sm.Logit(y_a, xareal_a)
    result = logit_model.fit()
    print(result.summary())
    predicted = result.predict(xareal_a)
    return predicted, xareal_a


def boot_fn(x_a, y_a):
    all_dataset = hstack((x_a,y_a))
    n = all_dataset.shape[0]
    index = arange(n)
    #print(index.shape)
    index_and_const = empty((n,2))
    index_and_const[:,0] = index
    index_and_const[:,1] = 1
    #print(index_and_const, index_and_const.shape)

    data_and_index = hstack((all_dataset, index_and_const))
    #print(data_and_index, data_and_index.shape)
    y_default = y_a[:,0]
    clf = GaussianNB()
    clf.fit(data_and_index, y_default)
    probs = clf.predict_proba(data_and_index)
    print(probs)
    #predicted, xareal_a = use_sm(x_a, y_a)
    #boot_fn(x_a)
    return data_and_index, y_default, probs

def main():
```

```
    x_a, y_a = data_loader("Default.csv")
    #use_sm(x_a, y_a)
    dist = norm(loc=2, scale=4)

    data = dist.rvs(size=10, random_state=rng)
    std_true = dist.std()

    print(std_true)
    std_sample = np.std(data)
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
    print(std_sample)
    raise SystemExit
    data_and_index, y_default, probs = boot_fn(x_a, y_a)
    #logit_model = sm.Logit(probs, y_default)
    #print(probs.shape, data_and_index.shape)
    #result = logit_model.fit()
    #print(result.summary())
    glmmodel = sm.GLM(probs, data_and_index)
    result = glmmodel.fit
    print(result.summary())

if __name__ == '__main__':
    main()
```

ch5p8.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import CubicSpline
from scipy.interpolate import splrep, BSpline
import statsmodels.api as sm
from sklearn.model_selection import LeaveOneOut

def data_loader():
    #random.seed(1)

    rng = np.random.default_rng(100)
    x_v = rng.normal(size = 100)
    y_v = (x_v) - (2 * x_v**2) + (rng.normal(size = 100))
    print(x_v.shape, y_v.shape)

    x_a = empty((100,5), dtype=float64)
    x_a[:,0] = 1
    x_a[:,1] = x_v
    x_a[:,2] = (x_v)**2
    x_a[:,3] = (x_v)**3
    x_a[:,4] = (x_v)**4
    #print("x_v", x_v)
    #print("x_a", x_a)
    return x_a, x_v, y_v

def data_scatterplot(x_v,y_v):
    plt.scatter(x_v,y_v)
    plt.show()

def line_lin_fit(x_a, y_v):
    #y_v = X@B
    b_v = linalg.pinv(x_a[:,0:2])@y_v
    print(b_v)

def line_loocv_fit(x_a, y_v):
    loo = LeaveOneOut()
    loo.get_n_splits(x_a)
    degree_v = arange(1,5)

    result_l = []

    for degree in degree_v:

        for train_i_v, test_i_v in loo.split(x_a):
            xtrain_a = x_a[train_i_v,:degree+1]
            ytrain_v = y_v[train_i_v]
            b_v = linalg.pinv(xtrain_a)@ytrain_v
            #print(b_v)

            yfit_v = x_a[:,:degree+1] @ b_v
            mse = ((y_v - yfit_v)**2).mean()
            result_l.append((b_v, mse))
            #print(mse)
    return result_l


def quad_lin_fit(x_a, y_v):
    #y_v = X@B
    b_v = linalg.pinv(x_a[:,0:3])@y_v
    print(b_v)
def cubic_lin_fit(x_a, y_v):
    #y_v = X@B
    b_v = linalg.pinv(x_a[:,0:4])@y_v
    print(b_v)
def xtofour_lin_fit(x_a, y_v):
    #y_v = X@B
    b_v = linalg.pinv(x_a[:,0:5])@y_v
    print(b_v)
def pvalue(x_a,y_v,name=''):
    if name:
        print(f'\n\n----------------------- {name} -------------------')
        model = sm.OLS(y_v, x_a)
        results = model.fit()
        print(results.summary())
    #Using stats models to get p value even though I did my own regression
def main():
    data_loader()
    x_a, x_v,  y_v = data_loader()
    #data_scatterplot(x_a, y_v)
    line_lin_fit(x_a, y_v)
    quad_lin_fit(x_a, y_v)
    cubic_lin_fit(x_a, y_v)
    xtofour_lin_fit(x_a, y_v)
    print("This seperates normal and LOOCV")
    mod_mse = line_loocv_fit(x_a, y_v)
    #smse_l = sorted(mod_mse, key = lambda x_t: x_t[1])
    #That sorted all the models, and found the one and its coefficents that produced the lowest mean squared error value
    #smse_l = sorted(mod_mse, key = lambda x_t: x_t[1]+x_t[0].shape!=2 *100000)
    #This one aimed at finding the linear model with the lowest mean squared error value. This was done by using the kornicer delta.
    #If the shape of the first term (where we had all the coeffients) was not 2 (!= is not equal ) (intercept and slope) it would increase the MSE
    #By 100,000 which means it wouldnt show up cause it is sorted by decreasing MSE

    for degree in range(1,5):
        #now changed it a bit so that it loops and prints what i need for all of them
        smse_l = sorted(mod_mse, key = lambda x_t: x_t[1]+(x_t[0].shape[0]!=degree+1)*100000)

```

```
        print('Degree', degree, smse_l[0])
    #print(mod_mse_l[:10])
    #print(mod_mse_l[100:110])


if __name__ == '__main__':
        main()
```

ch5p9.py

```python
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from sklearn.utils import resample
from scipy import stats
from scipy.stats import bootstrap
def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7,8,9,10,11,12,13), delimiter=',')

    return data_a

def pop_mean(data_a):
    #Want population mean of medv
    medv_v = data_a[:,13]
    muhat = medv_v.mean()
    #print(muhat)
    #print(medv_v)
    return medv_v, muhat

def stand_error_muhat(medv_v, muhat):
    #Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the
    # number of observations.
    stdmuhat = medv_v.std()
    n = medv_v.shape[0]

    stand_err_muhat = stdmuhat/(n**0.5)
    print(stand_err_muhat)
    return stand_err_muhat

def newboostrapstderror(medv_v):
    #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
    #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
    #https://www.statology.org/bootstrapping-in-python/
    #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977

    nbootstrap = 1000
    bootstrapmeans = []
    for _ in range(nbootstrap):
        bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
        bootstrap_sample_mean = mean(bootstrap_sample)
        bootstrapmeans.append(bootstrap_sample_mean)

    standard_error = np.std(bootstrapmeans)

    print("Standard Error of µˆusing Bootstrap:", standard_error)
    return standard_error
def newboostrapstderror_median(medv_v):
    #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
    #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
    #https://www.statology.org/bootstrapping-in-python/
    #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977

    nbootstrap = 1000
    bootstrapmedian = []
    for _ in range(nbootstrap):
        bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
        bootstrap_sample_median = median(bootstrap_sample)
        bootstrapmedian.append(bootstrap_sample_median)

    standard_error = std(bootstrapmedian)

    print("Standard Error of mu hat median using Bootstrap:", standard_error)
    return standard_error
def newbootstrapstderror_tenpercen(medv_v):
    #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
    #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
    #https://www.statology.org/bootstrapping-in-python/
    #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977

    nbootstrap = 1000
    bootstrappercen = []
    for _ in range(nbootstrap):
        bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
        bootstrap_sample_percen = percentile(bootstrap_sample, 10)
        bootstrappercen.append(bootstrap_sample_percen)

    standard_error = std(bootstrappercen)

    print("Standard Error of mu hat 0.1 using Bootstrap:", standard_error)

def muhat_median(data_a):
    medv_v = data_a[:,13]
    muhatmed = median(medv_v)
    return muhatmed
def main():
    data_a = data_loader("Boston.csv")
    medv_v, muhat = pop_mean(data_a)
```

```python
    stand_error_muhat(medv_v, muhat)
    standard_error = newboostrapstderror(medv_v)
    standard_error_median = newboostrapstderror_median(medv_v)
    print("standard error of median", standard_error_median)
    print(f'Con Int: [{muhat - 2*standard_error}, {muhat + 2*standard_error}')
    muhatmed = muhat_median(data_a)
    print(muhatmed)
    tenth_percen = percentile(medv_v, 10)
    print("Tenth Percentile (µ0.1) of medv:", tenth_percen)
    standard_error_percentile = newbootstrapstderror_tenpercen(medv_v)
    print(standard_error_percentile)
if __name__ == '__main__':
    main()
```