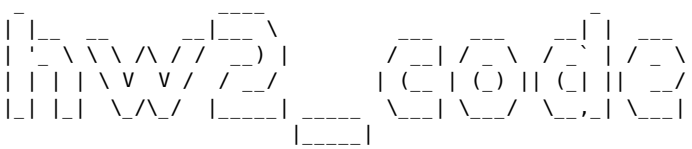

Stanford/ 1
├── .~lock.USArrests.csv# 1
├── ch12prob9.py 2
├── ch3prob9.py 5
├── ch3q14.py 8
├── ch4p13.py 10
├── ch5p2.py 14
├── ch5p5.py 15
├── ch5p6.py 17
├── ch5p8.py 19
├── ch5p9.py 21
├── dendogram.py 23
└── hw1graphs.py 25

.~lock.USArrests.csv#

```
1 ,adam,ahlan,07.07.2023 02:20,file:///home/adam/.config/libreoffice/4;
```

ch12prob9.py

```
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


#np.genfromtxt("Auto.csv", delimiter=",")
#with open("Auto.csv", "r") as f:
#    print(f.read())

#data_a = loadtxt('Auto.csv',skiprows=1, usecols=(0,3), delimiter=',')
#print(data_a)

#CHAPTER 3 QUESTION 9

def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')

    #mpg        cylinders       displacement    horsepower      weight  acceleration    year    origin  name


    #print(data_a.shape)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]

    n, m = data_a.shape
    pred_a = data_a[:, 1:]

    x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
    x_a[:,:-1] =  pred_a
    x_a[:,-1] = 1
    name_l = ['mpg',    'cylinders',    'displacement', 'horsepower',   'weight',       'acceleration', 'year', 'origin']

    #print(x_v.shape)

    y_v = data_a[:,0]

    return x_a, y_v, data_a, name_l

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def diag_plot(y_v, yfit_v):
    res_v = y_v - yfit_v
    i_v = yfit_v.argsort()
    yfit1_v = yfit_v[i_v]
    res1_v = res_v[i_v]
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.scatter(yfit1_v,res1_v)
    #spline = CubicSpline(yfit1_v, res1_v)
    #res2_v = spline(yfit1_v)
    tck = splrep(yfit1_v, res1_v, s=3500)
    res2_v = BSpline(*tck)(yfit1_v)
    #ax.plot(yfit1_v, res1_v)
    ax.plot(yfit1_v, res2_v)

```

```
    plt.show()

def transform1_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**2

    return hstack((p_a,x_a))

def transform2_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**0.5

    return hstack((p_a,x_a))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))




def lin_regression(x_a,y):
    #y_v = X@B
    b_v = linalg.pinv(x_a)@y
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v, data_a, name_l  = data_loader('Auto.csv')
    cor_a = corrcoef(data_a, rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)

    #print(x_a,y_v)
    #scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    b_v = lin_regression(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b_v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))

    #diag_plot(y_v, yfit_v)
    interaction_study(x_a, y_v, 'control')
    interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
    interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
    interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')
```

```
170
171     raise SystemExit
172
173     #print(yhat_v)
174 if __name__ == '__main__':
175         main()
176
177
178 # zeros, full, empty, array, arange, indexing, transpose
179
180 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
181 #
```

ch3prob9.py

```
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


#Adam Kainikara
#This code is for
#CHAPTER 3 QUESTION 9

def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')

    #mpg        cylinders       displacement    horsepower      weight  acceleration    year    origin  name


    #print(data_a.shape)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]

    n, m = data_a.shape
    pred_a = data_a[:, 1:]

    x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
    x_a[:,:-1] =  pred_a
    x_a[:,-1] = 1
    name_l = ['mpg',    'cylinders',    'displacement', 'horsepower',   'weight',       'acceleration', 'year', 'origin']

    #print(x_v.shape)

    y_v = data_a[:,0]

    return x_a, y_v, data_a, name_l

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def diag_plot(y_v, yfit_v):
    res_v = y_v - yfit_v
    i_v = yfit_v.argsort()
    yfit1_v = yfit_v[i_v]
    res1_v = res_v[i_v]
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.scatter(yfit1_v,res1_v)
    #spline = CubicSpline(yfit1_v, res1_v)
    #res2_v = spline(yfit1_v)
    tck = splrep(yfit1_v, res1_v, s=3500)
    res2_v = BSpline(*tck)(yfit1_v)
    #ax.plot(yfit1_v, res1_v)
    ax.plot(yfit1_v, res2_v)

    plt.show()

def transform1_reg(x_a):
    p_a = x_a[:,2:3]
```

```
    p_a = (p_a)**2

    return hstack((p_a,x_a))

def transform2_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = (p_a)**0.5

    return hstack((p_a,x_a))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v,name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r squared = ', r_square(y_v,yfit_v))




def lin_regression(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
    # y_v = X@B
    b_v = linalg.pinv(x_a)@y_v
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v, data_a, name_l  = data_loader('Auto.csv')
    cor_a = corrcoef(data_a, rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)

    #print(x_a,y_v)
    scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    b_v = lin_regression(x_a,y_v, name='Main Regression')
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b_v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))
    diag_plot(y_v, yfit_v)

    with np.printoptions(precision=2):
        interaction_study(x_a, y_v, 'control')
        interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
        interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
```

```
170         interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
171         interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
172         interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')
173 
174     raise SystemExit
175 
176     #print(yhat_v)
177 if __name__ == '__main__':
178         main()
179 
180 
181 # zeros, full, empty, array, arange, indexing, transpose
182 
183 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
184 #
```

ch3q14.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


#Adam Kainikara
#This code is for
#CHAPTER 3 QUESTION 14

def data_loader():
    random.seed(1)
    x1_v = random.uniform(size=100)
    x2_v = 0.5*x1_v + random.normal(size=100)/10
    y_v = 2+2*x1_v+0.3*x2_v+random.normal(size=100)
    x_a = empty((100,3), dtype=float64)
    x_a[:,0] = 1
    x_a[:,1] = x1_v
    x_a[:,2] = x2_v
    return x_a, y_v

def scatter_plot(x1_v,x2_v):
    plt.scatter(x1_v,x2_v)
    plt.show()

def rand_var_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v,name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r_squared = ', r_square(y_v,yfit_v))

def transform1_x1_only(x_a):
    #X1 and constant only
    return hstack((x_a[:,0:1],x_a[:,1:2]))


def transform1_x2_only(x_a):
    #X2 and constant only
    return hstack((x_a[:,0:1],x_a[:,2:]))

def transform3_reg(x_a):
    p_a = x_a[:,2:3]
    p_a = log((p_a))

    return hstack((p_a,x_a))

def transform4_reg(x_a):
    p_a = x_a[:,2:4]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))

def transform5_reg(x_a):
    p_a = x_a[:,2:5]
    p_a = product(p_a,axis=1,keepdims=True)

    return hstack((p_a,x_a))


def interaction_study(x_a, y_v, name):
    b_v = lin_regression(x_a,y_v, name=name)
    yfit_v = fitted_func(x_a, b_v)
    print(f'{name}: Coefficients {b_v=}')
    print(f'{name}: r_squared = ', r_square(y_v,yfit_v))




def lin_regression(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
```

```
    # y_v = X@B
    b_v = linalg.pinv(x_a)@y_v
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def main ():
    x_a, y_v = data_loader()
    b_v = lin regression(x_a,y_v,name='Main Regression')
    print('beta values', b_v)
    scatter_plot(x_a[:,1],x_a[:,2])
    cor_a = corrcoef(x_a[:,1:], rowvar=False)
    print(cor_a.shape)
    with np.printoptions(precision=4):
        print(cor_a)
    rand_var_study(x_a,y_v,'Control')
    rand_var_study(transform1_x1_only(x_a),y_v,'X1 Only')
    rand_var_study(transform1_x2_only(x_a),y_v,'X2 Only')

    x1_a = vstack((x_a,array([[1,0.1,0.8]])))
    y1_v = hstack((y_v,array([6])))
    rand_var_study(x_a,y_v,'New Control')
    rand_var_study(transform1_x1_only(x1_a),y1_v,'New X1 Only')
    rand_var_study(transform1_x2_only(x1_a),y1_v,'New X2 Only')

    raise SystemExit

    #print(x_a,y_v)
    #scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    yfit_v = fitted_func(x_a, b_v)
    i_v = abs(b v).argsort()[::-1]
    print(f'Coefficients {b_v=}')
    print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
    print("pred influence",[name_l[i] for i in i_v])
    print('r squared = ', r_square(y_v,yfit_v))

    #diag_plot(y_v, yfit_v)
    interaction_study(x_a, y_v, 'control')
    interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
    interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
    interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
    interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')


    #print(yhat v)
if __name__ == '__main__':
        main()


# zeros, full, empty, array, arange, indexing, transpose

#URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
#
```

ch4p13.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

#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)

```

```
103
104     plt.show()
105
106 def summaries(x_a):
107     print("Numerical summaries")
108     #min_x_a = x_a.min(axis = 0)
109     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)}')
110
111 def logi_reg(x_a, direction_v):
112     predictors_a = x_a[:,1:7]
113     #print(predictors_a)
114     #print(x_a.shape)
115     response_v = direction_v
116
117     #response_v = x_a[:,8]
118     #print(response_v)
119     logreg = LogisticRegression()
120     logreg.fit(predictors_a, response_v)
121     coefficients = logreg.coef_
122
123     print(coefficients)
124
125 def make_prediction(x_a, direction_v):
126     clf = LogisticRegression()
127     clf.fit(x_a, direction_v)
128     ypredict_v = clf.predict(x_a)
129     return ypredict_v
130
131
132 def compute_confusion_mat(ypredict_v, direction_v):
133     truey_v = direction_v
134     confu_mat = confusion_matrix(truey_v, ypredict_v)
135     print(confu_mat)
136     return confu_mat
137
138 def lda_prediction(x_v, y_v):
139     #This only served to help me write the code, I did it in main otherwise
140     clf = lda()
141     clf.fit(x_v, y_v)
142     ypredict_ldatrain_v = clf.predict(x_v)
143     ypredict_ldatest_v = clf.predict(x_v)
144     #Rembr to do the confusion matrix after
145     return ypredict_ldatrain_v, ypredict_ldatest_v
146     #What I ended up doing
147     clf = lda()
148     clf.fit(xtrain_v, ytrain_v)
149
150     ypredict_ldatrain_v = clf.predict(xtrain_v)
151     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used LDA")
152     compute_confusion_mat(ypredict_ldatrain_v, ytrain_v)
153
154     ypredict_ldatest_v = clf.predict(xtest_v)
155     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used LDA")
156     compute_confusion_mat(ypredict_ldatest_v, ytest_v)
157
158 def qda_prediction(x_v, y_v):
159     #This only served to help me write the code, I did it in main otherwise
160     clf = qda()
161     clf.fit(x_v, y_v)
162     ypredict_ldatrain_v = clf.predict(x_v)
163     ypredict_ldatest_v = clf.predict(x_v)
164     #Rembr to do the confusion matrix after
165     return ypredict_ldatrain_v, ypredict_ldatest_v
166
167     #What I ended up doing
168     clf = qda()
169     clf.fit(xtrain_v, ytrain_v)
170     ypredict qdatrain v = clf.predict(xtrain v)
171     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used QDA")
172     compute_confusion_mat(ypredict_qdatrain_v, ytrain_v)
173
174     ypredict qdatest v = clf.predict(xtest v)
175     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used QDA")
176     compute_confusion_mat(ypredict_qdatest_v, ytest_v)
177 def naiv_prediction(x_v, y_v):
178     #This onl served to elp me write the code, I did it in main otherwise
179     clf = GaussianNB()
180     clf.fit(x_v, y_v)
181     ypredict_nbtrain_v = clf.predict(x_v)
182     ypredict_nbtrest_v = clf.predict(x_v)
183     return ypredict_nbtrain_v, ypredict_nbtrest_v
184 def knn_prediction(x_v, y_v):
185     neigh = KNeighborsClassifier(n_neighbors=3)
186     neigh.fit(x_v, y_v)
187     ypredict_knntrain_v = neigh.predict(x_v)
188     ypredict_knntest_v = neigh.predict(x_v)
189     return ypredict_knntrain_v, ypredict_knntest_v
190
191 def main():
192     x_a, y_v = data_loader("Weekly.csv")
193
194     xtrain_v = x_a[:,1:2][:985]
195     xtest_v = x_a[:,1:2][985:]
196
197     ytrain_v = y_v[:985]
198     ytest_v = y_v[985:]
199
200
201     '''
202     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.
203     Then do the confusion matrix
204     '''
205
206     '''
207     This first part (below) is fitting the training data and then predicitng the y value based on the training data. I
208     t then computes the confusion matrix based on the training data
209     '''
```

```
210
211
212     clf = LogisticRegression()
213     clf.fit(xtrain_v, ytrain_v)
214
215
216     ytrain_pred_v = clf.predict(xtrain_v)
217     print("Confusion Matrix of the training data with Lag2 as the only predictor")
218     compute_confusion_mat(ytrain_pred_v, ytrain_v)
219
220
221     '''
222     This second part (below) is getting the predicited y value and computing
223     the confusion matrix based on the fit found earlier and the test data.
224     '''
225
226     ytest_pred_v = clf.predict(xtest_v)
227     print("Confusion Matrix of the test data with Lag2 as the only predictor")
228     compute_confusion_mat(ytest_pred_v, ytest_v)
229
230     """
231     ================================================================================================================================================
232     NOW DOING QDA
233     """
234     clf = lda()
235     clf.fit(xtrain_v, ytrain_v)
236     ypredict ldatrain v = clf.predict(xtrain v)
237     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used LDA")
238     compute_confusion_mat(ypredict_ldatrain_v, ytrain_v)
239
240     ypredict ldatest v = clf.predict(xtest v)
241     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used LDA")
242     compute_confusion_mat(ypredict_ldatest_v, ytest_v)
243
244     """
245     ================================================================================================================================================
246     NOW DOING QDA
247     """
248     clf = qda()
249     clf.fit(xtrain_v, ytrain_v)
250     ypredict qdatrain v = clf.predict(xtrain v)
251     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used QDA")
252     compute_confusion_mat(ypredict_qdatrain_v, ytrain_v)
253
254     ypredict qdatest v = clf.predict(xtest v)
255     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used QDA")
256     compute_confusion_mat(ypredict_qdatest_v, ytest_v)
257     """
258     ================================================================================================================================================
259     NOW DOING NAIVE BAEES
260     """
261     clf = GaussianNB()
262     clf.fit(xtrain_v, ytrain_v)
263     ypredict nbtrain v = clf.predict(xtrain v)
264     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used NAIVE BAYES")
265     compute_confusion_mat(ypredict_nbtrain_v, ytrain_v)
266
267     ypredict_nbtest_v = clf.predict(xtest_v)
268     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used NAIVE BAYES")
269     compute_confusion_mat(ypredict_nbtest_v, ytest_v)
270
271     """
272     ================================================================================================================================================
273     NOW DOING KNN
274     """
275     neigh = KNeighborsClassifier(n_neighbors=3)
276     neigh.fit(xtrain_v, ytrain_v)
277     ypredict knntrain v = neigh.predict(xtrain v)
278     print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used KNN")
279     compute_confusion_mat(ypredict_knntrain_v, ytrain_v)
280
281     ypredict knntest v = neigh.predict(xtest v)
282     print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used KNN")
283     compute_confusion_mat(ypredict_knntest_v, ytest_v)
284
285     """
286     ================================================================================================================================================
287     NOW DOING LDA WITH A TWIST
288     """
289     xtrain2_v = x_a[:,1:3][:985]
290     xtest2_v = x_a[:,1:3][985:]
291     ytrain_v = y_v[:985]
292     ytest_v = y_v[985:]
293     clf = lda()
294     clf.fit(xtrain2_v, ytrain_v)
295     ypredict ldatrain2 v = clf.predict(xtrain2 v)
296     print("Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors but instead used LDA")
297     compute_confusion_mat(ypredict_ldatrain2_v, ytrain_v)
298
299     ypredict ldatest2 v = clf.predict(xtest2 v)
300     print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used LDA")
301     compute_confusion_mat(ypredict_ldatest2_v, ytest_v)
302     """
303     ================================================================================================================================================
304     NOW DOING QDA WITH A TWIST
305     """
306     clf = qda()
307     clf.fit(xtrain2_v, ytrain_v)
308     ypredict_qdatrain2_v = clf.predict(xtrain2_v)
309     print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictors but instead used QDA")
310     compute_confusion_mat(ypredict_qdatrain2_v, ytrain_v)
311
312     ypredict_qdatest2_v = clf.predict(xtest2_v)
313     print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used QDA")
314     compute_confusion_mat(ypredict_qdatest2_v, ytest_v)
315     """
316     ================================================================================================================================================
```

```
    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

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

j = 1  # jth observation index
num_samples = 20000  # maximum value of n

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

```
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

```
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)

```

```
100         print('Degree', degree, smse_l[0])
101     #print(mod_mse_l[:10])
102     #print(mod_mse_l[100:110])
103 
104 
105 if __name__ == '__main__':
106         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)
```

```
    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()
```

dendogram.py

```
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.interpolate import splrep, BSpline
from scipy.cluster.hierarchy import complete, fcluster
from scipy.cluster import hierarchy
from scipy.spatial.distance import pdist

#Chapter 12 question 9




def data_loader(fname):
    x_a = loadtxt(fname,skiprows=1, usecols=(1,2,3,4), delimiter=',')


    state_v = loadtxt(fname, skiprows=1, usecols=(0), delimiter=',', dtype=str)



    print(state_v)
    #print(data_a[0:,1].shape)
    # We want it in the form  of Y = XB
    # Where Y is the response variable
    # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
    # B is the coeeficents (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]



    return x_a, state_v

def dendo_construct(x_a, state_v):
    n = state_v.shape[0]

    def label_func(id):
        if id < n:
            return state_v[id]
        return f'{id}'

    d_a = pdist(x_a)
    print(d_a.shape)
    print(d_a)
    z_a = complete(d_a)
    print(z_a.shape)
    print(z_a)
    plt.figure()
    dn = hierarchy.dendrogram(z_a, leaf_label_func=label_func)
    plt.show()

def z_score(x_a):
    m_v = x_a.mean(axis=0)
    std_v = x_a.std(axis=0)
    z_a = (x_a - m_v)/std_v
    return z_a

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    fig.savefig('ch12_9.pdf')
    plt.show()

def main ():
    x_a, state_v  = data_loader('USArrests.csv')
    dendo_construct(x_a, state_v)
    dendo_construct(z_score(x_a), state_v)
    raise SystemExit
```

```
87      raise SystemExit
88
89      #print(yhat_v)
90  if __name__ == '__main__':
91          main()
92
93
94  # zeros, full, empty, array, arange, indexing, transpose
95
96  #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
97  #https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html#scipy.cluster.hierarchy.dendrogram
```

hw1graphs.py

```
from numpy import *
import matplotlib.pyplot as plt
from matplotlib import patheffects
from statistics import linear_regression
from scipy.interpolate import CubicSpline


#THIS CODE IS FOR THE 5 THINGS GRAPH


fig, ax = plt.subplots(figsize=(15, 15))

#x x^2 constant
def powxy(x, y):
    return [x**2, x, 1, y]
def powxy2(x, y):
    return [x**-3, x**-2, x**-1, 1, y]
def startdata():
    pass
    data_a = array([
        powxy(0,4),
        #powxy(11,90),
        powxy(15,170),
        powxy(20,400)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_a = data_a[:,:-1]
    Y_a = data_a[:,-1]
    print(X_a.shape)
    print(Y_a.shape)
    Xinv_a = linalg.pinv(X_a)
    coefs = Xinv_a @ Y_a
    print(coefs)

    x_v = linspace(0.5,20,400)
    y_v = coefs[0] * (x_v **2) + coefs[1] * (x_v ** 1) + coefs[2]

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y_v)
    fig[0, 0].plot(x_v, y_v, label='var')
    ax.legend()

    plt.show()



def startdata_cubic():

    data_a = array([
        (0,4),
        (5,20),
        (15,140),
        (20,400)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='variance')
    ax.legend()


def bias_cubic():

    data_a = array([
        (0,400),
        (5,140),
        (15,20),
```

```
        (20,4)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='squared bias')

    ax.legend()

def trainerror_cubic():

    data_a = array([
        (0,340),
        (5,210),
        (10,110),
        (15,60),
        (20,15)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='training error')

    ax.legend()

def testerror_cubic():

    data_a = array([
        (0,350),
        (5,250),
        (10,140),
        (15,210),
        (20,340)
        ])
   # yvalue_start = array([
    #    powxy[4]
   # ])
    X_v = data_a[:,0]
    Y_v = data_a[:,1]
    spline = CubicSpline(X_v, Y_v)


    x1_v = linspace(0.5,20,400)
    y1_v = spline(x1_v)

    #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y1_v)
    ax.plot(x1_v, y1_v, label='test error')

    ax.legend()

def irrerror():
    ax.axhline(y=200, xmin = 0.05, xmax = 0.95, label='irreducible error')
    ax.legend()
def bias():
    data_a = array([
        powxy2(2,25),
        powxy2(3,16),
        powxy2(4,9),
        powxy2(5,4)
        ])
   # yvalue_start = array([
    #    powxy[4]
```

```
   # ])
    X_a = data_a[:,:-1]
    Y_a = data_a[:,-1]
    print(X_a.shape)
    print(Y_a.shape)
    Xinv_a = linalg.pinv(X_a)
    coefs = Xinv_a @ Y_a
    print(coefs)

    x_v = linspace(1,20,400)
    y_v = coefs[0] * (x_v **3) + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
    print(y_v)
    ax.plot(x_v, y_v, label='bias')
    ax.legend()

    plt.show()
def main():
    plt.xlabel("flexibility")
    plt.ylabel("value")
    startdata_cubic()
    bias_cubic()
    trainerror_cubic()
    testerror_cubic()
    irrerror()
    plt.show()
    #startdata()
    #bias()
if __name__ == '__main__':
        main()

#y_v = (coefs[0] * x_v) + ((coefs[1]) * 2) + coefs[3]

#https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html
```