

Stanford

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

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8
9
10 #np.genfromtxt("Auto.csv", delimiter=",")
11 #with open("Auto.csv", "r") as f:
12 #    print(f.read())
13
14 #data_a = loadtxt('Auto.csv',skiprows=1, usecols=(0,3), delimiter=',')
15 #print(data_a)
16
17 #CHAPTER 3 QUESTION 9
18
19 def data_loader(fname):
20     data_a = loadtxt(fname,skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')
21
22     #mpg          cylinders      displacement    horsepower      weight  acceleration   year    origin    name
23
24
25     #print(data_a.shape)
26     #print(data_a[0:,1].shape)
27     # We want it in the form of Y = XB
28     # Where Y is the response variable
29     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
30     # B is the coefficients (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]
31
32     n, m = data_a.shape
33     pred_a = data_a[:, 1:]
34
35     x_a = empty([n,pred_a.shape[1]+1], dtype=float64)
36     x_a[:, :-1] = pred_a
37     x_a[:, -1] = 1
38     name_l = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin']
39
40     #print(x_v.shape)
41
42     y_v = data_a[:,0]
43
44     return x_a, y_v, data_a, name_l
45
46 def scatter_matrix(data_a, name_l):
47     n, p = data_a.shape
48     fig, axs = plt.subplots(4, 7)
49     ax_l = list(axs.flat)
50     mpl.rcParams['figure.autolayout'] = True
51     font = {'family' : 'normal',
52             'weight' : 'bold',
53             'size' : 10}
54
55     #mpl.rc('font', **font)
56     for i in range(p):
57         for j in range(i+1,p):
58             print(i, j, name_l[i], name_l[j])
59             x_v = data_a[:,i]
60             y_v = data_a[:,j]
61             ax = ax_l.pop(0)
62             ax.scatter(x_v, y_v, s=2**2)
63             title = f'{name_l[i]} vs {name_l[j]}'
64             ax.set_title(title[:25])
65
66     plt.tight_layout()
67     fig.savefig('ch12_9.pdf')
68     plt.show()
69
70 def diag_plot(y_v, yfit_v):
71     res_v = y_v - yfit_v
72     i_v = yfit_v.argsort()
73     yfit1_v = yfit_v[i_v]
74     res1_v = res_v[i_v]
75     fig, ax = plt.subplots(figsize=(15, 15))
76     ax.scatter(yfit1_v, res1_v)
77     #spline = CubicSpline(yfit1_v, res1_v)
78     #res2_v = spline(yfit1_v)
79     tck = splrep(yfit1_v, res1_v, s=3500)
80     res2_v = BSpline(*tck)(yfit1_v)
81     #ax.plot(yfit1_v, res1_v)
82     ax.plot(yfit1_v, res2_v)
```

```

83     plt.show()
84
85     def transform1_reg(x_a):
86         p_a = x_a[:,2:3]
87         p_a = (p_a)**2
88
89         return hstack((p_a,x_a))
90
91     def transform2_reg(x_a):
92         p_a = x_a[:,2:3]
93         p_a = (p_a)**0.5
94
95         return hstack((p_a,x_a))
96
97     def transform3_reg(x_a):
98         p_a = x_a[:,2:3]
99         p_a = log((p_a))
100
101         return hstack((p_a,x_a))
102
103     def transform4_reg(x_a):
104         p_a = x_a[:,2:4]
105         p_a = product(p_a,axis=1,keepdims=True)
106
107         return hstack((p_a,x_a))
108
109     def transform5_reg(x_a):
110         p_a = x_a[:,2:5]
111         p_a = product(p_a,axis=1,keepdims=True)
112
113         return hstack((p_a,x_a))
114
115
116     def interaction_study(x_a, y_v, name):
117         b_v = lin_regression(x_a,y_v)
118         yfit_v = fitted_func(x_a, b_v)
119         print(f'{name}: Coefficients {b_v=}')
120         print(f'{name}: r squared = ', r_square(y_v,yfit_v))
121
122
123
124
125     def lin_regression(x_a,y):
126         #y_v = X@B
127         b_v = linalg.pinv(x_a)@y
128         #print(b)
129         return b_v
130
131     def fitted_func(x_a,b_v):
132         yfit_v = x_a@b_v
133         # with np.printoptions(precision=2):
134         #     print(f'predicted mpg of cars {yfit_v=}')
135         return yfit_v
136
137     def r_square(y_v,yfit_v):
138         # This function is to find the r squared value
139         # This will be calculated by doing 1 - variance of (actual - predicted)/variance of actual
140         rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
141
142         return rsq
143
144     def main ():
145         x_a, y_v, data_a, name_l = data_loader('Auto.csv')
146         cor_a = corrcoef(data_a, rowvar=False)
147         print(cor_a.shape)
148         with np.printoptions(precision=4):
149             print(cor_a)
150
151         #print(x_a,y_v)
152         #scatter_matrix(data_a, name_l)
153
154         #print(x_a,y_v)
155         b_v = lin_regression(x_a,y_v)
156         yfit_v = fitted_func(x_a, b_v)
157         i_v = abs(b_v).argsort()[::-1]
158         print(f'Coefficients {b_v=}')
159         print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
160         print("pred_influence",[name_l[i] for i in i_v])
161         print('r squared = ', r_square(y_v,yfit_v))
162
163         #diag_plot(y_v, yfit_v)
164         interaction_study(x_a, y_v, 'control')
165         interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
166         interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
167         interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
168         interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
169         interaction_study(transform5_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

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8 import statsmodels.api as sm
9
10
11 #Adam Kainikara
12 #This code is for
13 #CHAPTER 3 QUESTION 9
14
15 def data_loader(fname):
16     data_a = loadtxt(fname, skiprows=1, usecols=(0,1,2,3,4,5,6,7), delimiter=',')
17
18     #mpg      cylinders      displacement      horsepower      weight      acceleration      year      origin      name
19
20
21     #print(data_a.shape)
22     #print(data_a[0:,1].shape)
23     # We want it in the form of Y = XB
24     # Where Y is the response variable
25     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
26     # B is the coefficients (slope and intercept) that we are trying to solve for     year_v = data_a[:,1]
27
28     n, m = data_a.shape
29     pred_a = data_a[:, 1:]
30
31     x_a = empty([n, pred_a.shape[1]+1], dtype=float64)
32     x_a[:, :-1] = pred_a
33     x_a[:, -1] = 1
34     name_l = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year', 'origin']
35
36     #print(x_v.shape)
37
38     y_v = data_a[:, 0]
39
40     return x_a, y_v, data_a, name_l
41
42 def scatter_matrix(data_a, name_l):
43     n, p = data_a.shape
44     fig, axs = plt.subplots(4, 7)
45     ax_l = list(axs.flat)
46     mpl.rcParams['figure.autolayout'] = True
47     font = {'family' : 'normal',
48             'weight' : 'bold',
49             'size' : 10}
50
51     #mpl.rc('font', **font)
52     for i in range(p):
53         for j in range(i+1, p):
54             print(i, j, name_l[i], name_l[j])
55             x_v = data_a[:, i]
56             y_v = data_a[:, j]
57             ax = ax_l.pop(0)
58             ax.scatter(x_v, y_v, s=2**2)
59             title = f'{name_l[i]} vs {name_l[j]}'
60             ax.set_title(title[:25])
61
62     plt.tight_layout()
63     fig.savefig('ch12_9.pdf')
64     plt.show()
65
66 def diag_plot(y_v, yfit_v):
67     res_v = y_v - yfit_v
68     i_v = yfit_v.argsort()
69     yfit1_v = yfit_v[i_v]
70     res1_v = res_v[i_v]
71     fig, ax = plt.subplots(figsize=(15, 15))
72     ax.scatter(yfit1_v, res1_v)
73     #spline = CubicSpline(yfit1_v, res1_v)
74     #res2_v = spline(yfit1_v)
75     tck = splrep(yfit1_v, res1_v, s=3500)
76     res2_v = BSpline(*tck)(yfit1_v)
77     #ax.plot(yfit1_v, res1_v)
78     ax.plot(yfit1_v, res2_v)
79
80     plt.show()
81
82 def transform1_reg(x_a):
83     p_a = x_a[:, 2:3]
```

```

83     p_a = (p_a)**2
84
85     return hstack((p_a,x_a))
86
87 def transform2_reg(x_a):
88     p_a = x_a[:,2:3]
89     p_a = (p_a)**0.5
90
91     return hstack((p_a,x_a))
92
93 def transform3_reg(x_a):
94     p_a = x_a[:,2:3]
95     p_a = log((p_a))
96
97     return hstack((p_a,x_a))
98
99 def transform4_reg(x_a):
100    p_a = x_a[:,2:4]
101    p_a = product(p_a,axis=1,keepdims=True)
102
103    return hstack((p_a,x_a))
104
105 def transform5_reg(x_a):
106    p_a = x_a[:,2:5]
107    p_a = product(p_a,axis=1,keepdims=True)
108
109    return hstack((p_a,x_a))
110
111
112 def interaction_study(x_a, y_v, name):
113     b_v = lin_regression(x_a,y_v,name=name)
114     yfit_v = fitted_func(x_a, b_v)
115     print(f'{name}: Coefficients {b_v=}')
116     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
117
118
119
120
121 def lin_regression(x_a,y_v,name=''):
122     if name:
123         print(f'\n\n----- {name} -----')
124         model = sm.OLS(y_v, x_a)
125         results = model.fit()
126         print(results.summary())
127         #Using stats models to get p value even though I did my own regression
128         # y_v = X@B
129         b_v = linalg.pinv(x_a)@y_v
130         #print(b)
131         return b_v
132
133 def fitted_func(x_a,b_v):
134     yfit_v = x_a@b_v
135     # with np.printoptions(precision=2):
136     #     print(f'predicted mpg of cars {yfit_v=}')
137     return yfit_v
138
139 def r_square(y_v,yfit_v):
140     # This function is to find the r squared value
141     # This will be calculated by doing 1 - variance of (actual - predicated)/variance of actual
142     rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
143
144     return rsq
145
146 def main ():
147     x_a, y_v, data_a, name_l = data_loader('Auto.csv')
148     cor_a = corrcoef(data_a, rowvar=False)
149     print(cor_a.shape)
150     with np.printoptions(precision=4):
151         print(cor_a)
152
153     #print(x_a,y_v)
154     scatter_matrix(data_a, name_l)
155
156     #print(x_a,y_v)
157     b_v = lin_regression(x_a,y_v, name='Main Regression')
158     yfit_v = fitted_func(x_a, b_v)
159     i_v = abs(b_v).argsort()[::-1]
160     print(f'Coefficients {b_v=}')
161     print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
162     print("pred influence",[name_l[i] for i in i_v])
163     print('r squared = ', r_square(y_v,yfit_v))
164     diag_plot(y_v, yfit_v)
165
166     with np.printoptions(precision=2):
167         interaction_study(x_a, y_v, 'control')
168         interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
169         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

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8 import statsmodels.api as sm
9
10
11 #Adam Kainikara
12 #This code is for
13 #CHAPTER 3 QUESTION 14
14
15 def data_loader():
16     random.seed(1)
17     x1_v = random.uniform(size=100)
18     x2_v = 0.5*x1_v + random.normal(size=100)/10
19     y_v = 2+2*x1_v+0.3*x2_v+random.normal(size=100)
20     x_a = empty((100,3), dtype=float64)
21     x_a[:,0] = 1
22     x_a[:,1] = x1_v
23     x_a[:,2] = x2_v
24     return x_a, y_v
25
26 def scatter_plot(x1_v,x2_v):
27     plt.scatter(x1_v,x2_v)
28     plt.show()
29
30 def rand_var_study(x_a, y_v, name):
31     b_v = lin_regression(x_a,y_v,name=name)
32     yfit_v = fitted_func(x_a, b_v)
33     print(f'{name}: Coefficients {b_v=}')
34     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
35
36 def transform1_x1_only(x_a):
37     #X1 and constant only
38     return hstack((x_a[:,0:1],x_a[:,1:2]))
39
40
41 def transform1_x2_only(x_a):
42     #X2 and constant only
43     return hstack((x_a[:,0:1],x_a[:,2:]))
44
45 def transform3_reg(x_a):
46     p_a = x_a[:,2:3]
47     p_a = log((p_a))
48
49     return hstack((p_a,x_a))
50
51 def transform4_reg(x_a):
52     p_a = x_a[:,2:4]
53     p_a = product(p_a,axis=1,keepdims=True)
54
55     return hstack((p_a,x_a))
56
57 def transform5_reg(x_a):
58     p_a = x_a[:,2:5]
59     p_a = product(p_a,axis=1,keepdims=True)
60
61     return hstack((p_a,x_a))
62
63
64 def interaction_study(x_a, y_v, name):
65     b_v = lin_regression(x_a,y_v, name=name)
66     yfit_v = fitted_func(x_a, b_v)
67     print(f'{name}: Coefficients {b_v=}')
68     print(f'{name}: r squared = ', r_square(y_v,yfit_v))
69
70
71
72
73 def lin_regression(x_a,y_v,name=''):
74     if name:
75         print(f'\n\n----- {name} -----')
76         model = sm.OLS(y_v, x_a)
77         results = model.fit()
78         print(results.summary())
79     #Using stats models to get p value even though I did my own regression
```

```

80     # y_v = X@B
81     b_v = linalg.pinv(x_a)@y_v
82     #print(b)
83     return b_v
84
85 def fitted_func(x_a,b_v):
86     yfit_v = x_a@b_v
87     # with np.printoptions(precision=2):
88     #     print(f'predicted mpg of cars {yfit_v=}')
89     return yfit_v
90
91 def r_square(y_v,yfit_v):
92     # This function is to find the r squared value
93     # This will be calculated by doing 1 - variance of (actual - predicted)/variance of actual
94     rsq = 1 - (var(y_v-yfit_v))/(var(y_v))
95
96     return rsq
97
98 def main ():
99     x_a, y_v = data_loader()
100     b_v = lin_regression(x_a,y_v,name='Main Regression')
101     print('beta values', b_v)
102     scatter_plot(x_a[:,1],x_a[:,2])
103     cor_a = corcoef(x_a[:,1:], rowvar=False)
104     print(cor_a.shape)
105     with np.printoptions(precision=4):
106         print(cor_a)
107     rand_var_study(x_a,y_v,'Control')
108     rand_var_study(transform1_x1_only(x_a),y_v,'X1 Only')
109     rand_var_study(transform1_x2_only(x_a),y_v,'X2 Only')
110
111     x1_a = vstack((x_a,array([[1,0.1,0.8]])))
112     y1_v = hstack((y_v,array([6])))
113     rand_var_study(x_a,y_v,'New Control')
114     rand_var_study(transform1_x1_only(x1_a),y1_v,'New X1 Only')
115     rand_var_study(transform1_x2_only(x1_a),y1_v,'New X2 Only')
116
117     raise SystemExit
118
119     #print(x_a,y_v)
120     #scatter_matrix(data_a, name_l)
121
122     #print(x_a,y_v)
123     yfit_v = fitted_func(x_a, b_v)
124     i_v = abs(b_v).argsort()[::-1]
125     print(f'Coefficients {b_v=}')
126     print('Coefficients:', [(name, b) for name, b in zip(name_l, b_v)])
127     print("pred influence", [name_l[i] for i in i_v])
128     print('r squared = ', r_square(y_v,yfit_v))
129
130     #diag_plot(y_v, yfit_v)
131     interaction_study(x_a, y_v, 'control')
132     interaction_study(transform1_reg(x_a),y_v, 'Horse Power Squared')
133     interaction_study(transform2_reg(x_a),y_v, 'Horse Power Square Root')
134     interaction_study(transform3_reg(x_a),y_v, 'Horse Power Log')
135     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight')
136     interaction_study(transform4_reg(x_a),y_v, 'Horse Power * Weight * Acceleration')
137
138
139     #print(yhat_v)
140 if __name__ == '__main__':
141     main()
142
143
144 # zeros, full, empty, array, arange, indexing, transpose
145
146 #URLs used: https://docs.scipy.org/doc/scipy/tutorial/interpolate/smoothing_splines.html
147 #

```

```

1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6
7 import sys
8 import matplotlib
9
10 import matplotlib.pyplot as plt
11 import numpy
12 from sklearn.datasets import load_iris
13 from sklearn.linear_model import LogisticRegression
14 from sklearn.metrics import confusion_matrix
15 #from sklearn.metrics import accuracy_score #works
16 from matplotlib.ticker import FormatStrFormatter
17 import statsmodels.api as sm
18
19 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
20 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
21 from sklearn.neighbors import KNeighborsClassifier
22 from sklearn.naive_bayes import GaussianNB
23
24 #Adam Kainikara
25 #This code is for
26 #CHAPTER 4 QUESTION 13
27 #THIS IS FOR PROBLEM 5
28 #OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202
29
30
31 def data_loader(fname):
32     data_a = loadtxt(fname, skiprows=1, usecols=(1,2,3,4,5,6,7), delimiter=',')
33     direction_v = loadtxt(fname, skiprows=1, usecols=(8), delimiter=',', dtype=str)
34
35     return data_a, direction_v
36
37 def summaries0(x_a):
38     print("Numerical summaries")
39     for i in range(x_a.shape[1]):
40         column = x_a[:, i]
41         print(f'Column {i + 1}:')
42         print(f' - Min: {min(column)}')
43         print(f' - Max: {max(column)}')
44         print(f' - Mean: {mean(column)}')
45         print(f' - Median: {median(column)}')
46         print(f' - Stand Dev: {std(column)}')
47     print(x_a.shape[1])
48
49     summary = empty((8,5))
50     storage_l = []
51
52     for i in range(x_a.shape[1]):
53         column = x_a[:, i]
54         summary = [min(column), max(column), mean(column), median(column), std(column)]
55         print(summary)
56         storage_l.append(summary)
57     return np.array(storage_l)
58
59 def calc_summary(x_a) -> dict[str, ndarray]:
60     data_d = {
61         "min": x_a.min(axis = 0),
62         "max": x_a.max(axis = 0),
63         "mean": x_a.mean(axis = 0),
64         "stand dev": x_a.std(axis = 0),
65         "median": median(x_a, axis = 0)
66     }
67     return data_d
68
69 def print_summary(data_d: dict[str, ndarray]):
70     for k,v in data_d.items():
71         print(f'{k}: ', v)
72
73 def find_p_values(x_a, direction_v):
74     #y_v = direction_v
75     #log_reg = sm.Logit(y_v, ex x_a).fit()
76
77     # Extract the p-values
78     #print(log_reg.summary())
79     pass
80
81 def plot_summary(data_d):
82     data_d = {'min': data_d['min'], 'max': data_d['max'], 'mean': data_d['mean'], 'stan dev': data_d['stand dev'], 'median': data_d['median']}
83
84     nvar = len(data_d["min"])
85     print(data_d)
86     w = 0.25
87     stride = w + 0.05
88     initial_change = 0
89     labels = ("Min", "Max", "Mean", "Stan Dev", "Median")
90     x_v = arange(nvar) * stride + 0
91     print(x_v)
92
93     fig, ax = plt.subplots()
94
95     for attribute, measurement in data_d.items():
96         #shift = w * initial_change
97         rectangle = ax.bar(x_v, measurement, width=w, label = attribute)
98         x_v += nvar * stride
99         ax.bar_label(rectangle, fmt = '%.02f', rotation = 45, padding=2)
100         #initial change += 0.25
101         #ax.set_xticks(x_v, labels)

```

```

103 plt.show()
104
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_qdatrain_v = clf.predict(x_v)
163     ypredict_qdatest_v = clf.predict(x_v)
164     #Rembr to do the confusion matrix after
165     return ypredict_qdatrain_v, ypredict_qdatest_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
178 def naiv_prediction(x_v, y_v):
179     #This onl served to elp me write the code, I did it in main otherwise
180     clf = GaussianNB()
181     clf.fit(x_v, y_v)
182     ypredict_nbtrain_v = clf.predict(x_v)
183     ypredict_nbtest_v = clf.predict(x_v)
184     return ypredict_nbtrain_v, ypredict_nbtest_v
185
186 def knn_prediction(x_v, y_v):
187     neigh = KNeighborsClassifier(n_neighbors=3)
188     neigh.fit(x_v, y_v)
189     ypredict_knntrain_v = neigh.predict(x_v)
190     ypredict_knntest_v = neigh.predict(x_v)
191     return ypredict_knntrain_v, ypredict_knntest_v
192
193 def main():
194     x_a, y_v = data_loader("Weekly.csv")
195
196     xtrain_v = x_a[:,1:2][:985]
197     xtest_v = x_a[:,1:2][985:]
198
199     ytrain_v = y_v[:985]
200     ytest_v = y_v[985:]
201
202     '''
203     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.
204     Then do the confusion matrix
205     '''
206
207     '''
208     This first part (below) is fitting the training data and then predictng the y value based on the training data. I
209     t then computes the confusion matrix based on the training data
210     '''

```

```

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 predicted 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 =====
260 NOW DOING NAIVE BAEES
261 '''
262 clf = GaussianNB()
263 clf.fit(xtrain_v, ytrain_v)
264 ypredict_nbtrain_v = clf.predict(xtrain_v)
265 print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used NAIVE BAYES")
266 compute_confusion_mat(ypredict_nbtrain_v, ytrain_v)
267
268 ypredict_nbtest_v = clf.predict(xtest_v)
269 print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used NAIVE BAYES")
270 compute_confusion_mat(ypredict_nbtest_v, ytest_v)
271
272
273 =====
274 NOW DOING KNN
275 '''
276 neigh = KNeighborsClassifier(n_neighbors=3)
277 neigh.fit(xtrain_v, ytrain_v)
278 ypredict_kntrain_v = neigh.predict(xtrain_v)
279 print("Confusion Matrix of the training data with Lag2 as the only predictor but instead used KNN")
280 compute_confusion_mat(ypredict_kntrain_v, ytrain_v)
281
282 ypredict_kntest_v = neigh.predict(xtest_v)
283 print("Confusion Matrix of the test data with Lag2 as the only predictor but instead used KNN")
284 compute_confusion_mat(ypredict_kntest_v, ytest_v)
285
286
287 =====
288 NOW DOING LDA WITH A TWIST
289 '''
290 xtrain2_v = x_a[:,1:3][:985]
291 xtest2_v = x_a[:,1:3][985:]
292 ytrain_v = y_v[:985]
293 ytest_v = y_v[985:]
294 clf = lda()
295 clf.fit(xtrain2_v, ytrain_v)
296 ypredict_ldatrain2_v = clf.predict(xtrain2_v)
297 print("Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors but instead used LDA")
298 compute_confusion_mat(ypredict_ldatrain2_v, ytrain_v)
299
300 ypredict_ldatest2_v = clf.predict(xtest2_v)
301 print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used LDA")
302 compute_confusion_mat(ypredict_ldatest2_v, ytest_v)
303
304
305 =====
306 NOW DOING QDA WITH A TWIST
307 '''
308 clf = qda()
309 clf.fit(xtrain2_v, ytrain_v)
310 ypredict_qdatrain2_v = clf.predict(xtrain2_v)
311 print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictors but instead used QDA")
312 compute_confusion_mat(ypredict_qdatrain2_v, ytrain_v)
313
314 ypredict_qdatest2_v = clf.predict(xtest2_v)
315 print("Confusion Matrix of the test data with Lag2 and Lag3 as the only predictors but instead used QDA")
316 compute_confusion_mat(ypredict_qdatest2_v, ytest_v)
317
318 =====

```

```

317 NOW DOING NAIVE BAYES WITH A TWIST
318 """
319 clf = GaussianNB()
320 clf.fit(xtrain2_v, ytrain_v)
321 ypredict_nbtrain2_v = clf.predict(xtrain2_v)
322 print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictor but instead used NAIVE BAYES")
323 compute_confusion_mat(ypredict_nbtrain2_v, ytrain_v)
324
325 ypredict_nbtest2_v = clf.predict(xtest2_v)
326 print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor but instead used NAIVE BAYES")
327 compute_confusion_mat(ypredict_nbtest2_v, ytest_v)
328
329
330
331 =====
332 NOW DOING KNN WITH A TWIST
333 """
334 neigh = KNeighborsClassifier(n_neighbors=7)
335 neigh.fit(xtrain2_v, ytrain_v)
336 ypredict_kntrain2_v = neigh.predict(xtrain2_v)
337 print("Confusion Matrix of the training data with Lag2 and Lag 3 as the only predictor but instead used KNN")
338 compute_confusion_mat(ypredict_kntrain2_v, ytrain_v)
339
340 ypredict_kntest2_v = neigh.predict(xtest2_v)
341 print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor but instead used KNN")
342 compute_confusion_mat(ypredict_kntest2_v, ytest_v)
343
344 clf = LogisticRegression()
345 clf.fit(xtrain2_v, ytrain_v)
346
347
348 ytrain_pred2_v = clf.predict(xtrain2_v)
349 print("Confusion Matrix of the training data with Lag2 and Lag3 as the only predictors")
350 compute_confusion_mat(ytrain_pred2_v, ytrain_v)
351
352
353 ytest_pred2_v = clf.predict(xtest2_v)
354 print("Confusion Matrix of the test data with Lag2 and Lag 3 as the only predictor")
355 compute_confusion_mat(ytest_pred2_v, ytest_v)
356
357
358
359
360 #print(xtest_a)
361 #print(ytrain_v)
362
363 #ytest_v = y_v[:ntest]
364 #xtrain_v = x_a[:ntest:]
365
366 #confusion_pract()
367 #print(x_a)
368 #logi_reg(x_a, y_v)
369 #find_p_values(x_a, direction_v)
370 #summaries(x_a)
371 summary = summaries(x_a)
372 #calc_summary(x_a)
373 data_dict = calc_summary(x_a)
374 print_summary(data_dict)
375 plot_summary(data_dict)
376
377
378 clf = LogisticRegression()
379 clf.fit(xtrain_v, ytrain_v)
380 ytrain_pred_v = clf.predict(xtrain_v)
381 compute_confusion_mat(ytrain_pred_v, ytrain_v)
382 ytest_pred_v = clf.predict(xtest_v)
383 compute_confusion_mat(ytest_pred_v, ytest_v)
384 print('Done----')
385
386 make_prediction(x_a, y_v)
387 ypredict_v = make_prediction(x_a, y_v)
388 print(ypredict_v)
389 compute_confusion_mat(ypredict_v, y_v)
390 #print(type(summary))
391 #print(summary.shape)
392 #plot_summaries(summary)
393
394
395
396 #Some Links I used
397 #https://scikit-learn.org/stable/auto_examples/classification/plot_lda_qda.html#sphx-glr-auto-examples-classification-plot-lda-qda-py
398 #https://stackoverflow.com/questions/46775155/importerror-no-module-named-sklearn-lda
399
400 if __name__ == '__main__':
401     main()

```

ch5p2.py

```
1 from numpy import *
2 import matplotlib.pyplot as plt
3
4 j = 1 # jth observation index
5 num_samples = 20000 # maximum value of n
6
7 n_values = arange(1, num_samples + 1)
8 probabilities = []
9
10 for n in n_values:
11     sample = random.choice(range(n), size=n, replace=True)
12     probability = mean(sample == j)
13     probabilities.append(probability)
14
15 # Creating a scatter plot
16 plt.scatter(n_values, probabilities, s=5)
17 plt.xlabel('n')
18 plt.ylabel(f'Probability of {j}th observation in bootstrap sample')
19 plt.title('Probability of jth observation in bootstrap sample for different values of n')
20 plt.grid(True)
21 plt.show()
```


ch5p5.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6
7 import sys
8 import matplotlib
9
10 import matplotlib.pyplot as plt
11 import numpy
12 from sklearn.datasets import load_iris
13 from sklearn.linear_model import LogisticRegression
14 from sklearn.metrics import confusion_matrix
15 #from sklearn.metrics import accuracy_score #works
16 from matplotlib.ticker import FormatStrFormatter
17 import statsmodels.api as sm
18
19 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
20 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
21 from sklearn.neighbors import KNeighborsClassifier
22 from sklearn.naive_bayes import GaussianNB
23 from sklearn.metrics import accuracy_score
24
25
26 #Adam Kainikara
27 #This code is for
28 #CHAPTER 5 QUESTION 5
29 #THIS IS PROBLEM 7
30 #OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202
31
32 def data_loader(fname):
33     num_data_a = loadtxt(fname, skiprows=1, usecols=(2,3), delimiter=',')
34     defa_student_a = loadtxt(fname, skiprows=1, usecols=(0,1), delimiter=',', dtype=str)
35
36     return num_data_a, defa_student_a
37
38 def logi_reg(x_a, y_a):
39     predictors_a = x_a
40     response_v = y_a[:,0]
41     #print(predictors_a, response_v)
42     logreg = LogisticRegression()
43     logreg.fit(predictors_a, response_v)
44     coefficients = logreg.coef_
45     print(coefficients)
46 def new_logi_reg(x_a, y_a):
47     pass
48
49 def compute_confusion_mat(ypredict_v, direction_v):
50     truey_v = direction_v
51     confu_mat = confusion_matrix(truey_v, ypredict_v)
52     print(confu_mat)
53     return confu_mat
54
55 def main():
56     x_a, y_a = data_loader("Default.csv")
57
58
59     ydefault_v = y_a[:,0]
60     print(y_a[:,1])
61     #ysudent = [1 if x == "yes" else 0 for x in x_a[:,1]]
62     ystudent = [1 if x == "Yes" else 0 for x in y_a[:,1]]
63     #print(ystudent)
64
65
66     #print(x_a)
67     student_a = array(ystudent)
68     #print(student_a.shape)
69     xall_a = array([x_a[:,0], x_a[:,1], student_a])
70     realx_a = transpose(xall_a)
71     print(realx_a.shape)
72     print(realx_a)
73
74     xalltrain_a = realx_a[:5000]
75     xallvalid_a = realx_a[5000:]
76     print(xalltrain_a, xallvalid_a)
77     ytrain_v = ydefault_v[:5000]
78     yvalid_v = ydefault_v[5000:]
79     clf = LogisticRegression()
```

```

80     clf.fit(xalltrain_a, ytrain_v)
81     ytrain_pred_v = clf.predict(xalltrain_a)
82     yvalid_pred_v = clf.predict(xallvalid_a)
83     coefficients = clf.coef_
84     print(coefficients)
85     compute_confusion_mat(ytrain_pred_v, yvalid_pred_v)
86     raise SystemExit
87
88     xall_a = ([x_a],[ysudent])
89     print(xall_a)
90     #logi_reg(x_a,y_a)
91
92
93     xtrain_a = x_a[:5000]
94     xvalid_a = x_a[5000:]
95     ytrain_v = ydefault_v[:5000]
96     yvalid_v = ydefault_v[5000:]
97
98
99
100    clf = LogisticRegression()
101    clf.fit(xtrain_a, ytrain_v)
102    ytrain_pred_v = clf.predict(xtrain_a)
103    yvalid_pred_v = clf.predict(xvalid_a)
104    coefficients = clf.coef_
105    print(coefficients)
106    #compute_confusion_mat(ytrain_pred_v, yvalid_pred_v)
107
108
109    clf = GaussianNB()
110    clf.fit(xtrain_a, ytrain_v)
111    posterior_probs = clf.predict_proba(xvalid_a)
112    predictions = (posterior_probs > 0.5)
113    print(predictions)
114    accuracy = accuracy_score(yvalid_v, predictions)
115    print("Accuracy:", accuracy)
116    #print(xtrain_a, ytrain_v)
117    raise SystemExit
118
119    ytrain_v = y_v[:985]
120    #ytest_v = y_v[985:]
121
122
123
124
125 if __name__ == '__main__':
126     main()

```

ch5p6.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6
7 import sys
8 import matplotlib
9
10 import matplotlib.pyplot as plt
11 import numpy
12 from sklearn.datasets import load_iris
13 from sklearn.linear_model import LogisticRegression
14 from sklearn.metrics import confusion_matrix
15 #from sklearn.metrics import accuracy_score #works
16 from matplotlib.ticker import FormatStrFormatter
17 import statsmodels.api as sm
18
19 from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as lda
20 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as qda
21 from sklearn.neighbors import KNeighborsClassifier
22 from sklearn.naive_bayes import GaussianNB
23 from sklearn.metrics import accuracy_score
24
25 import statsmodels.api as sm
26 rng = np.random.default_rng()
27 from scipy.stats import norm
28 #Adam Kainikara
29 #This code is for
30 #CHAPTER 5 QUESTION 6
31 #THIS IS PROBLEM 8
32 #OF HOMEWORK 2 FOR STANFORD SUMMER SESSION STATS 202
33
34 def data_loader(fname):
35     num_data_a = loadtxt(fname, skiprows=1, usecols=(2,3), delimiter=',')
36     defa_a = loadtxt(fname, skiprows=1, usecols=(0,1), delimiter=',', dtype=str)
37     ydefault = [1 if x == "Yes" else 0 for x in defa_a[:,0]]
38     ystudent = [1 if x == "Yes" else 0 for x in defa_a[:,1]]
39
40     default_a = transpose(array((ydefault, ystudent)))
41     #print(default_a)
42     return num_data_a, default_a
43
44 def use_sm(x_a, y_a):
45
46     b = ones((10000,1))
47     xareal_a = hstack((x_a,b))
48     print(y_a.dtype)
49
50     print(xareal_a)
51     logit_model = sm.Logit(y_a, xareal_a)
52     result = logit_model.fit()
53     print(result.summary())
54     predicted = result.predict(xareal_a)
55     return predicted, xareal_a
56
57
58 def boot_fn(x_a, y_a):
59     all_dataset = hstack((x_a,y_a))
60     n = all_dataset.shape[0]
61     index = arange(n)
62     #print(index.shape)
63     index_and_const = empty((n,2))
64     index_and_const[:,0] = index
65     index_and_const[:,1] = 1
66     #print(index_and_const, index_and_const.shape)
67
68     data_and_index = hstack((all_dataset, index_and_const))
69     #print(data_and_index, data_and_index.shape)
70     y_default = y_a[:,0]
71     clf = GaussianNB()
72     clf.fit(data_and_index, y_default)
73     probs = clf.predict_proba(data_and_index)
74     print(probs)
75     #predicted, xareal_a = use_sm(x_a, y_a)
76     #boot_fn(x_a)
77     return data_and_index, y_default, probs
78
79 def main():
```

```

80     x_a, y_a = data_loader("Default.csv")
81     #use_sm(x_a, y_a)
82     dist = norm(loc=2, scale=4)
83
84     data = dist.rvs(size=10, random_state=rng)
85     std_true = dist.std()
86
87     print(std_true)
88     std_sample = np.std(data)
89 # https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
90     print(std_sample)
91     raise SystemExit
92     data_and_index, y_default, probs = boot_fn(x_a, y_a)
93     #logit_model = sm.Logit(probs, y_default)
94     #print(probs.shape, data_and_index.shape)
95     #result = logit_model.fit()
96     #print(result.summary())
97     glmmodel = sm.GLM(probs, data_and_index)
98     result = glmmodel.fit
99     print(result.summary())
100
101 if __name__ == '__main__':
102     main()

```

```

1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy import stats
6 from scipy.interpolate import CubicSpline
7 from scipy.interpolate import splrep, BSpline
8 import statsmodels.api as sm
9 from sklearn.model_selection import LeaveOneOut
10
11 def data_loader():
12     #random.seed(1)
13
14     rng = np.random.default_rng(100)
15     x_v = rng.normal(size = 100)
16     y_v = (x_v) - (2 * x_v**2) + (rng.normal(size = 100))
17     print(x_v.shape, y_v.shape)
18
19     x_a = empty((100,5), dtype=float64)
20     x_a[:,0] = 1
21     x_a[:,1] = x_v
22     x_a[:,2] = (x_v)**2
23     x_a[:,3] = (x_v)**3
24     x_a[:,4] = (x_v)**4
25     #print("x_v", x_v)
26     #print("x_a", x_a)
27     return x_a, x_v, y_v
28
29 def data_scatterplot(x_v,y_v):
30     plt.scatter(x_v,y_v)
31     plt.show()
32
33 def line_lin_fit(x_a, y_v):
34     #y_v = X@B
35     b_v = linalg.pinv(x_a[:,0:2])@y_v
36     print(b_v)
37
38 def line_loocv_fit(x_a, y_v):
39     loo = LeaveOneOut()
40     loo.get_n_splits(x_a)
41     degree_v = arange(1,5)
42
43     result_l = []
44
45     for degree in degree_v:
46
47         for train_i_v, test_i_v in loo.split(x_a):
48             xtrain_a = x_a[train_i_v,:degree+1]
49             ytrain_v = y_v[train_i_v]
50             b_v = linalg.pinv(xtrain_a)@ytrain_v
51             #print(b_v)
52
53             yfit_v = x_a[:,degree+1:] @ b_v
54             mse = ((y_v - yfit_v)**2).mean()
55             result_l.append((b_v, mse))
56             #print(mse)
57     return result_l
58
59
60 def quad_lin_fit(x_a, y_v):
61     #y_v = X@B
62     b_v = linalg.pinv(x_a[:,0:3])@y_v
63     print(b_v)
64 def cubic_lin_fit(x_a, y_v):
65     #y_v = X@B
66     b_v = linalg.pinv(x_a[:,0:4])@y_v
67     print(b_v)
68 def xtofour_lin_fit(x_a, y_v):
69     #y_v = X@B
70     b_v = linalg.pinv(x_a[:,0:5])@y_v
71     print(b_v)
72 def pvalue(x_a,y_v,name=''):
73     if name:
74         print(f'\n\n----- {name} -----')
75         model = sm.OLS(y_v, x_a)
76         results = model.fit()
77         print(results.summary())
78     #Using stats models to get p value even though I did my own regression
79 def main():
80     data_loader()
81     x_a, x_v, y_v = data_loader()
82     #data_scatterplot(x_a, y_v)
83     line_lin_fit(x_a, y_v)
84     quad_lin_fit(x_a, y_v)
85     cubic_lin_fit(x_a, y_v)
86     xtofour_lin_fit(x_a, y_v)
87     print("This separates normal and LOOCV")
88     mod_mse = line_loocv_fit(x_a, y_v)
89     #smse_l = sorted(mod_mse, key = lambda x_t: x_t[1])
90     #That sorted all the models, and found the one and its coefficients that produced the lowest mean squared error value
91     #smse_l = sorted(mod_mse, key = lambda x_t: x_t[1]+x_t[0].shape[0]*100000)
92     #This one aimed at finding the linear model with the lowest mean squared error value. This was done by using the kornicer delta.
93     #If the shape of the first term (where we had all the coefficients) was not 2 (!= is not equal ) (intercept and slope) it would increase the MSE
94     #By 100,000 which means it wouldnt show up cause it is sorted by decreasing MSE
95
96     for degree in range(1,5):
97         #now changed it a bit so that it loops and prints what i need for all of them
98         smse_l = sorted(mod_mse, key = lambda x_t: x_t[1]+(x_t[0].shape[0]!=degree+1)*100000)
99

```

```
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

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 import numpy as np
6 from sklearn.utils import resample
7 from scipy import stats
8 from scipy.stats import bootstrap
9 def data_loader(fname):
10     data_a = loadtxt(fname, skiprows=1, usecols=(0,1,2,3,4,5,6,7,8,9,10,11,12,13), delimiter=',')
11
12     return data_a
13
14 def pop_mean(data_a):
15     #Want population mean of medv
16     medv_v = data_a[:,13]
17     muhat = medv_v.mean()
18     #print(muhat)
19     #print(medv_v)
20     return medv_v, muhat
21
22 def stand_error_muhat(medv_v, muhat):
23     #Hint: We can compute the standard error of the sample mean by dividing the sample standard deviation by the square root of the
24     # number of observations.
25     stdmuhat = medv_v.std()
26     n = medv_v.shape[0]
27
28     stand_err_muhat = stdmuhat/(n**0.5)
29     print(stand_err_muhat)
30     return stand_err_muhat
31
32 def newbootstrapstderror(medv_v):
33     #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
34     #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
35     #https://www.statology.org/bootstrapping-in-python/
36     #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977
37
38     nbootstrap = 1000
39     bootstrapmeans = []
40     for _ in range(nbootstrap):
41         bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
42         bootstrap_sample_mean = mean(bootstrap_sample)
43         bootstrapmeans.append(bootstrap_sample_mean)
44
45     standard_error = np.std(bootstrapmeans)
46
47     print("Standard Error of  $\mu$  using Bootstrap:", standard_error)
48     return standard_error
49
50 def newbootstrapstderror_median(medv_v):
51     #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
52     #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
53     #https://www.statology.org/bootstrapping-in-python/
54     #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977
55
56     nbootstrap = 1000
57     bootstrapmedian = []
58     for _ in range(nbootstrap):
59         bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
60         bootstrap_sample_median = median(bootstrap_sample)
61         bootstrapmedian.append(bootstrap_sample_median)
62
63     standard_error = std(bootstrapmedian)
64
65     print("Standard Error of  $\mu$  hat median using Bootstrap:", standard_error)
66     return standard_error
67
68 def newbootstrapstderror_tenpercen(medv_v):
69     #I am not sure if i coded a method of bootstrap correctly. I referenced the following websites
70     #https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.bootstrap.html
71     #https://www.statology.org/bootstrapping-in-python/
72     #https://medium.com/swlh/bootstrap-sampling-using-pythons-numpy-85822d868977
73
74     nbootstrap = 1000
75     bootstrappercen = []
76     for _ in range(nbootstrap):
77         bootstrap_sample = random.choice(medv_v, size=len(medv_v), replace=True)
78         bootstrap_sample_percen = percentile(bootstrap_sample, 10)
79         bootstrappercen.append(bootstrap_sample_percen)
80
81     standard_error = std(bootstrappercen)
82
83     print("Standard Error of  $\mu$  hat 0.1 using Bootstrap:", standard_error)
84
85 def muhat_median(data_a):
86     medv_v = data_a[:,13]
87     muhatmed = median(medv_v)
88     return muhatmed
89
90 def main():
91     data_a = data_loader("Boston.csv")
92     medv_v, muhat = pop_mean(data_a)
```

```

90     stand_error_muhat(medv_v, muhat)
91     standard_error = newbootstrapstderror(medv_v)
92     standard_error_median = newbootstrapstderror_median(medv_v)
93     print("standard error of median", standard_error_median)
94     print(f'Con Int: [{muhat - 2*standard_error}, {muhat + 2*standard_error}']')
95     muhatmed = muhat_median(data_a)
96     print(muhatmed)
97     tenth_percen = percentile(medv_v, 10)
98     print("Tenth Percentile ( $\mu_{0.1}$ ) of medv:", tenth_percen)
99     standard_error_percentile = newbootstrapstderror_tenpercen(medv_v)
100    print(standard_error_percentile)
101 if __name__ == '__main__':
102     main()

```


dendogram.py

```
1 from numpy import *
2 import numpy as np
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
5 from scipy.interpolate import splrep, BSpline
6 from scipy.cluster.hierarchy import complete, fcluster
7 from scipy.cluster import hierarchy
8 from scipy.spatial.distance import pdist
9
10 #Chapter 12 question 9
11
12
13
14
15
16 def data_loader(fname):
17     x_a = loadtxt(fname, skiprows=1, usecols=(1,2,3,4), delimiter=',')
18
19
20     state_v = loadtxt(fname, skiprows=1, usecols=(0), delimiter=',', dtype=str)
21
22
23
24     print(state_v)
25     #print(data_a[0:,1].shape)
26     # We want it in the form of Y = XB
27     # Where Y is the response variable
28     # Where X is an array with size nx2 where n is the predictor variable and the other column is a 1
29     # B is the coefficients (slope and intercept) that we are trying to solve for    year_v = data_a[:,1]
30
31
32
33     return x_a, state_v
34
35 def dendo_construct(x_a, state_v):
36     n = state_v.shape[0]
37
38     def label_func(id):
39         if id < n:
40             return state_v[id]
41         return f'{id}'
42
43     d_a = pdist(x_a)
44     print(d_a.shape)
45     print(d_a)
46     z_a = complete(d_a)
47     print(z_a.shape)
48     print(z_a)
49     plt.figure()
50     dn = hierarchy.dendrogram(z_a, leaf_label_func=label_func)
51     plt.show()
52
53 def z_score(x_a):
54     m_v = x_a.mean(axis=0)
55     std_v = x_a.std(axis=0)
56     z_a = (x_a - m_v)/std_v
57     return z_a
58
59 def scatter_matrix(data_a, name_l):
60     n, p = data_a.shape
61     fig, axs = plt.subplots(4, 7)
62     ax_l = list(axs.flat)
63     mpl.rcParams['figure.autolayout'] = True
64     font = {'family' : 'normal',
65            'weight' : 'bold',
66            'size' : 10}
67
68     #mpl.rc('font', **font)
69     for i in range(p):
70         for j in range(i+1,p):
71             print(i, j, name_l[i], name_l[j])
72             x_v = data_a[:,i]
73             y_v = data_a[:,j]
74             ax = ax_l.pop(0)
75             ax.scatter(x_v, y_v, s=2**2)
76             title = f'{name_l[i]} vs {name_l[j]}'
77             ax.set_title(title[:25])
78     plt.tight_layout()
79     fig.savefig('ch12_9.pdf')
80     plt.show()
81
82 def main():
83     x_a, state_v = data_loader('USArrests.csv')
84     dendo_construct(x_a, state_v)
85     dendo_construct(z_score(x_a), state_v)
86     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

```
1 from numpy import *
2 import matplotlib.pyplot as plt
3 from matplotlib import patheffects
4 from statistics import linear_regression
5 from scipy.interpolate import CubicSpline
6
7
8 #THIS CODE IS FOR THE 5 THINGS GRAPH
9
10
11 fig, ax = plt.subplots(figsize=(15, 15))
12
13 #x x^2 constant
14 def powxy(x, y):
15     return [x**2, x, 1, y]
16 def powxy2(x, y):
17     return [x**-3, x**-2, x**-1, 1, y]
18 def startdata():
19     pass
20     data_a = array([
21         powxy(0,4),
22         #powxy(11,90),
23         powxy(15,170),
24         powxy(20,400)
25     ])
26     # yvalue_start = array([
27     #     powxy[4]
28     # ])
29     X_a = data_a[:, :-1]
30     Y_a = data_a[:, -1]
31     print(X_a.shape)
32     print(Y_a.shape)
33     Xinv_a = linalg.pinv(X_a)
34     coefs = Xinv_a @ Y_a
35     print(coefs)
36
37     x_v = linspace(0.5, 20, 400)
38     y_v = coefs[0] * (x_v ** 2) + coefs[1] * (x_v ** 1) + coefs[2]
39
40     #y_v = coefs[0] * (x_v ** 3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
41     print(y_v)
42     fig[0, 0].plot(x_v, y_v, label='var')
43     ax.legend()
44
45     plt.show()
46
47
48
49 def startdata_cubic():
50
51     data_a = array([
52         (0,4),
53         (5,20),
54         (15,140),
55         (20,400)
56     ])
57     # yvalue_start = array([
58     #     powxy[4]
59     # ])
60     X_v = data_a[:, 0]
61     Y_v = data_a[:, 1]
62     spline = CubicSpline(X_v, Y_v)
63
64
65     x1_v = linspace(0.5, 20, 400)
66     y1_v = spline(x1_v)
67
68     #y_v = coefs[0] * (x_v ** 3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
69     print(y1_v)
70     ax.plot(x1_v, y1_v, label='variance')
71     ax.legend()
72
73
74 def bias_cubic():
75
76     data_a = array([
77         (0,400),
78         (5,140),
79         (15,20),
```

```

80         (20,4)
81     ])
82     # yvalue_start = array([
83     #     powxy[4]
84     # ])
85     X_v = data_a[:,0]
86     Y_v = data_a[:,1]
87     spline = CubicSpline(X_v, Y_v)
88
89
90     x1_v = linspace(0.5,20,400)
91     y1_v = spline(x1_v)
92
93     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
94     print(y1_v)
95     ax.plot(x1_v, y1_v, label='squared bias')
96
97     ax.legend()
98
99 def trainererror_cubic():
100
101     data_a = array([
102         (0,340),
103         (5,210),
104         (10,110),
105         (15,60),
106         (20,15)
107     ])
108     # yvalue_start = array([
109     #     powxy[4]
110     # ])
111     X_v = data_a[:,0]
112     Y_v = data_a[:,1]
113     spline = CubicSpline(X_v, Y_v)
114
115
116     x1_v = linspace(0.5,20,400)
117     y1_v = spline(x1_v)
118
119     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
120     print(y1_v)
121     ax.plot(x1_v, y1_v, label='training error')
122
123     ax.legend()
124
125 def testerror_cubic():
126
127     data_a = array([
128         (0,350),
129         (5,250),
130         (10,140),
131         (15,210),
132         (20,340)
133     ])
134     # yvalue_start = array([
135     #     powxy[4]
136     # ])
137     X_v = data_a[:,0]
138     Y_v = data_a[:,1]
139     spline = CubicSpline(X_v, Y_v)
140
141
142     x1_v = linspace(0.5,20,400)
143     y1_v = spline(x1_v)
144
145     #y_v = coefs[0] * (x_v **3 + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
146     print(y1_v)
147     ax.plot(x1_v, y1_v, label='test error')
148
149     ax.legend()
150
151 def irrerror():
152     ax.axhline(y=200, xmin = 0.05, xmax = 0.95, label='irreducible error')
153     ax.legend()
154 def bias():
155     data_a = array([
156         powxy2(2,25),
157         powxy2(3,16),
158         powxy2(4,9),
159         powxy2(5,4)
160     ])
161     # yvalue_start = array([
162     #     powxy[4]

```

```

163     # ])
164     X_a = data_a[:, :-1]
165     Y_a = data_a[:, -1]
166     print(X_a.shape)
167     print(Y_a.shape)
168     Xinv_a = linalg.pinv(X_a)
169     coefs = Xinv_a @ Y_a
170     print(coefs)
171
172     x_v = linspace(1, 20, 400)
173     y_v = coefs[0] * (x_v ** 3) + coefs[1] * (x_v ** 2) + coefs[2] * (x_v ) + coefs[3]
174     print(y_v)
175     ax.plot(x_v, y_v, label='bias')
176     ax.legend()
177
178     plt.show()
179     def main():
180         plt.xlabel("flexibility")
181         plt.ylabel("value")
182         startdata_cubic()
183         bias_cubic()
184         trainerror_cubic()
185         testerror_cubic()
186         irrerror()
187         plt.show()
188         #startdata()
189         #bias()
190     if __name__ == '__main__':
191         main()
192
193     #y_v = (coefs[0] * x_v) + ((coefs[1]) * 2) + coefs[3]
194
195     #https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.axhline.html

```