w6 - tanay

August 28, 2024

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

q1 Apply scikit learn model for Simple Linear regression using SGD of the given Salary_Data.csv dataset, and arrive at different values of B0, B1 and error for varying iterations. Plot the graph of epoch(X-axis) versus error(Y-axis)

[2]:
```
data = {
    "salary" : [1.7,2.4,2.3, 3.1, 3.7, 4.2, 4.4, 6.1, 5.4, 5.7, 6.4, 6.2],
    "experience" : [1.2, 1.5, 1.9, 2.2, 2.4, 2.5, 2.8, 3.1, 3.3, 3.7, 4.2, 4.4]
}
df = pd.DataFrame(data)
df
sal_df = df
```

[3]:
```
x = np.array(df["salary"]).reshape(-1,1)
y = np.array(df["experience"]).reshape(-1,1)
```

[ ]:
```

```

[4]:
```
model = LinearRegression()
model.fit(x, y)

print(model.coef_, model.intercept_)
```

```
[[0.57968648]] [0.27401481]
```

[5]:
```
predictions = model.predict(x)
```

[6]:
```
predictions
```

[6]:
```
array([[1.25948182],
       [1.66526236],
       [1.60729371],
       [2.07104289],
       [2.41885478],
```

```
       [2.70869802],
       [2.82463531],
       [3.81010233],
       [3.40432179],
       [3.57822774],
       [3.98400827],
       [3.86807098]])
```

[7]:
```python
epochs = 400
alpha = 0.001
n = df.shape[0]
b0,b1=0,0

epoch = []
errors = []

for i in range(epochs):
    for j in range(n):
        xi = df["experience"][j]
        yi = df["salary"][j]

        pi = b0 + b1*xi

        err = pi - xi

        b0 = b0 - alpha*err
        b1 = b1 - b1*alpha*err*df["experience"]

        if j==n-1:
            print('EPOCH: ', i)
            predictions = b0 + b1*df["experience"]

            mse = np.sum((predictions - df["salary"]) ** 2)
            epoch.append(i)
            errors.append(mse)

print(epoch)
print(errors)
```

```
EPOCH:  0
EPOCH:  1
EPOCH:  2
EPOCH:  3
EPOCH:  4
EPOCH:  5
EPOCH:  6
```

```
EPOCH:  7
EPOCH:  8
EPOCH:  9
EPOCH:  10
EPOCH:  11
EPOCH:  12
EPOCH:  13
EPOCH:  14
EPOCH:  15
EPOCH:  16
EPOCH:  17
EPOCH:  18
EPOCH:  19
EPOCH:  20
EPOCH:  21
EPOCH:  22
EPOCH:  23
EPOCH:  24
EPOCH:  25
EPOCH:  26
EPOCH:  27
EPOCH:  28
EPOCH:  29
EPOCH:  30
EPOCH:  31
EPOCH:  32
EPOCH:  33
EPOCH:  34
EPOCH:  35
EPOCH:  36
EPOCH:  37
EPOCH:  38
EPOCH:  39
EPOCH:  40
EPOCH:  41
EPOCH:  42
EPOCH:  43
EPOCH:  44
EPOCH:  45
EPOCH:  46
EPOCH:  47
EPOCH:  48
EPOCH:  49
EPOCH:  50
EPOCH:  51
EPOCH:  52
EPOCH:  53
EPOCH:  54
```

```
EPOCH:  55
EPOCH:  56
EPOCH:  57
EPOCH:  58
EPOCH:  59
EPOCH:  60
EPOCH:  61
EPOCH:  62
EPOCH:  63
EPOCH:  64
EPOCH:  65
EPOCH:  66
EPOCH:  67
EPOCH:  68
EPOCH:  69
EPOCH:  70
EPOCH:  71
EPOCH:  72
EPOCH:  73
EPOCH:  74
EPOCH:  75
EPOCH:  76
EPOCH:  77
EPOCH:  78
EPOCH:  79
EPOCH:  80
EPOCH:  81
EPOCH:  82
EPOCH:  83
EPOCH:  84
EPOCH:  85
EPOCH:  86
EPOCH:  87
EPOCH:  88
EPOCH:  89
EPOCH:  90
EPOCH:  91
EPOCH:  92
EPOCH:  93
EPOCH:  94
EPOCH:  95
EPOCH:  96
EPOCH:  97
EPOCH:  98
EPOCH:  99
EPOCH:  100
EPOCH:  101
EPOCH:  102
```

```
EPOCH:  103
EPOCH:  104
EPOCH:  105
EPOCH:  106
EPOCH:  107
EPOCH:  108
EPOCH:  109
EPOCH:  110
EPOCH:  111
EPOCH:  112
EPOCH:  113
EPOCH:  114
EPOCH:  115
EPOCH:  116
EPOCH:  117
EPOCH:  118
EPOCH:  119
EPOCH:  120
EPOCH:  121
EPOCH:  122
EPOCH:  123
EPOCH:  124
EPOCH:  125
EPOCH:  126
EPOCH:  127
EPOCH:  128
EPOCH:  129
EPOCH:  130
EPOCH:  131
EPOCH:  132
EPOCH:  133
EPOCH:  134
EPOCH:  135
EPOCH:  136
EPOCH:  137
EPOCH:  138
EPOCH:  139
EPOCH:  140
EPOCH:  141
EPOCH:  142
EPOCH:  143
EPOCH:  144
EPOCH:  145
EPOCH:  146
EPOCH:  147
EPOCH:  148
EPOCH:  149
EPOCH:  150
```

```
EPOCH:  151
EPOCH:  152
EPOCH:  153
EPOCH:  154
EPOCH:  155
EPOCH:  156
EPOCH:  157
EPOCH:  158
EPOCH:  159
EPOCH:  160
EPOCH:  161
EPOCH:  162
EPOCH:  163
EPOCH:  164
EPOCH:  165
EPOCH:  166
EPOCH:  167
EPOCH:  168
EPOCH:  169
EPOCH:  170
EPOCH:  171
EPOCH:  172
EPOCH:  173
EPOCH:  174
EPOCH:  175
EPOCH:  176
EPOCH:  177
EPOCH:  178
EPOCH:  179
EPOCH:  180
EPOCH:  181
EPOCH:  182
EPOCH:  183
EPOCH:  184
EPOCH:  185
EPOCH:  186
EPOCH:  187
EPOCH:  188
EPOCH:  189
EPOCH:  190
EPOCH:  191
EPOCH:  192
EPOCH:  193
EPOCH:  194
EPOCH:  195
EPOCH:  196
EPOCH:  197
EPOCH:  198
```

```
EPOCH:  199
EPOCH:  200
EPOCH:  201
EPOCH:  202
EPOCH:  203
EPOCH:  204
EPOCH:  205
EPOCH:  206
EPOCH:  207
EPOCH:  208
EPOCH:  209
EPOCH:  210
EPOCH:  211
EPOCH:  212
EPOCH:  213
EPOCH:  214
EPOCH:  215
EPOCH:  216
EPOCH:  217
EPOCH:  218
EPOCH:  219
EPOCH:  220
EPOCH:  221
EPOCH:  222
EPOCH:  223
EPOCH:  224
EPOCH:  225
EPOCH:  226
EPOCH:  227
EPOCH:  228
EPOCH:  229
EPOCH:  230
EPOCH:  231
EPOCH:  232
EPOCH:  233
EPOCH:  234
EPOCH:  235
EPOCH:  236
EPOCH:  237
EPOCH:  238
EPOCH:  239
EPOCH:  240
EPOCH:  241
EPOCH:  242
EPOCH:  243
EPOCH:  244
EPOCH:  245
EPOCH:  246
```

```
EPOCH:  247
EPOCH:  248
EPOCH:  249
EPOCH:  250
EPOCH:  251
EPOCH:  252
EPOCH:  253
EPOCH:  254
EPOCH:  255
EPOCH:  256
EPOCH:  257
EPOCH:  258
EPOCH:  259
EPOCH:  260
EPOCH:  261
EPOCH:  262
EPOCH:  263
EPOCH:  264
EPOCH:  265
EPOCH:  266
EPOCH:  267
EPOCH:  268
EPOCH:  269
EPOCH:  270
EPOCH:  271
EPOCH:  272
EPOCH:  273
EPOCH:  274
EPOCH:  275
EPOCH:  276
EPOCH:  277
EPOCH:  278
EPOCH:  279
EPOCH:  280
EPOCH:  281
EPOCH:  282
EPOCH:  283
EPOCH:  284
EPOCH:  285
EPOCH:  286
EPOCH:  287
EPOCH:  288
EPOCH:  289
EPOCH:  290
EPOCH:  291
EPOCH:  292
EPOCH:  293
EPOCH:  294
```

```
EPOCH:  295
EPOCH:  296
EPOCH:  297
EPOCH:  298
EPOCH:  299
EPOCH:  300
EPOCH:  301
EPOCH:  302
EPOCH:  303
EPOCH:  304
EPOCH:  305
EPOCH:  306
EPOCH:  307
EPOCH:  308
EPOCH:  309
EPOCH:  310
EPOCH:  311
EPOCH:  312
EPOCH:  313
EPOCH:  314
EPOCH:  315
EPOCH:  316
EPOCH:  317
EPOCH:  318
EPOCH:  319
EPOCH:  320
EPOCH:  321
EPOCH:  322
EPOCH:  323
EPOCH:  324
EPOCH:  325
EPOCH:  326
EPOCH:  327
EPOCH:  328
EPOCH:  329
EPOCH:  330
EPOCH:  331
EPOCH:  332
EPOCH:  333
EPOCH:  334
EPOCH:  335
EPOCH:  336
EPOCH:  337
EPOCH:  338
EPOCH:  339
EPOCH:  340
EPOCH:  341
EPOCH:  342
```

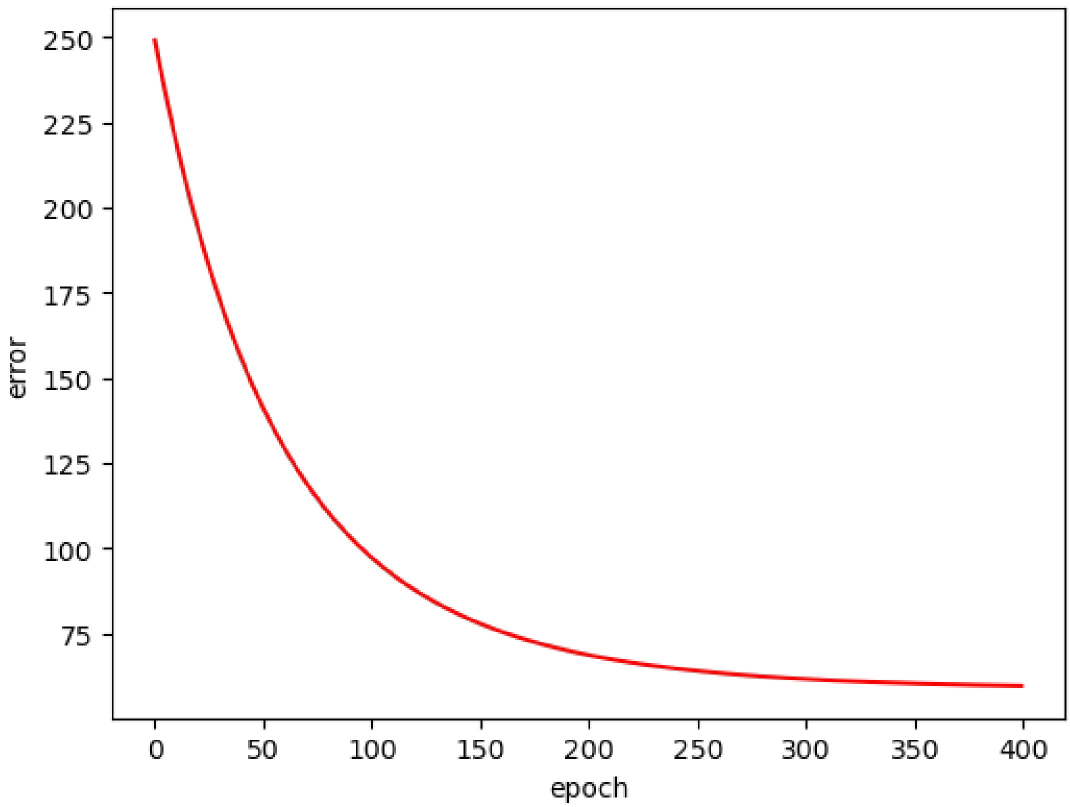
```
EPOCH:  343
EPOCH:  344
EPOCH:  345
EPOCH:  346
EPOCH:  347
EPOCH:  348
EPOCH:  349
EPOCH:  350
EPOCH:  351
EPOCH:  352
EPOCH:  353
EPOCH:  354
EPOCH:  355
EPOCH:  356
EPOCH:  357
EPOCH:  358
EPOCH:  359
EPOCH:  360
EPOCH:  361
EPOCH:  362
EPOCH:  363
EPOCH:  364
EPOCH:  365
EPOCH:  366
EPOCH:  367
EPOCH:  368
EPOCH:  369
EPOCH:  370
EPOCH:  371
EPOCH:  372
EPOCH:  373
EPOCH:  374
EPOCH:  375
EPOCH:  376
EPOCH:  377
EPOCH:  378
EPOCH:  379
EPOCH:  380
EPOCH:  381
EPOCH:  382
EPOCH:  383
EPOCH:  384
EPOCH:  385
EPOCH:  386
EPOCH:  387
EPOCH:  388
EPOCH:  389
EPOCH:  390
```

```
EPOCH:  391
EPOCH:  392
EPOCH:  393
EPOCH:  394
EPOCH:  395
EPOCH:  396
EPOCH:  397
EPOCH:  398
EPOCH:  399
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41,
42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116,
117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132,
133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148,
149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164,
165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180,
181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196,
197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228,
229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244,
245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260,
261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276,
277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292,
293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308,
309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340,
341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356,
357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372,
373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388,
389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399]
[249.10154775551416, 245.76941339645433, 242.50219428826725, 239.29851903666906,
236.1570467707678, 233.07646644290463, 230.05549614481995, 227.0928824397604,
224.18739971015293, 221.33784952047844, 218.54305999498902, 215.80188520991888,
213.113204599849, 210.47592237789286, 207.88896696937738, 205.35129045870383,
202.8618680490775, 200.41969753480453, 198.0237987858608, 195.6732132444444,
193.36700343323037, 191.1042524750529, 188.88406362374704, 186.7055598058877,
184.56788317317006, 182.4701946651818, 180.41167358232363, 178.3915171686396,
176.40894020432478, 174.46317460768358, 172.553469046317, 170.67908855732205,
168.8393141762928, 167.03344257491588, 165.26078570695975, 163.5206704624604,
161.812438329912, 160.1354450662744, 158.48906037461484, 156.87266758920498,
155.28566336789834, 153.72745739161815, 152.19747207078873, 150.69514225854837,
149.2199149705845, 147.77124911143656, 146.348615207115, 144.95149514388888,
143.57938191309782, 142.23177936184726, 140.90820194944942, 139.60817450947613,
138.33123201729165, 137.07691936293836, 135.84479112924902, 134.63441137506507,
133.44535342344008, 132.2771996547136, 131.12954130434, 130.00197826536285,
```

128.89411889542563, 127.80557982821279, 126.73598578921822, 125.68496941574065,
124.65217108100585, 123.63723872232123, 122.6398276731677, 121.65960049913733,
120.69622683762772, 119.7493832412045, 118.81875302454777, 117.90402611489766,
117.00489890591948, 116.12107411490639, 115.25226064324437, 114.3981734400619,
113.558533368991, 112.7330670779673, 111.92150687199835, 111.12359058883102,
110.339061477451, 109.56766807934852, 108.80916411248545, 108.06330835790251,
107.32986454890334, 106.6086012627574, 105.89929181486218, 105.20171415530814,
104.51565076779069, 103.84088857081427, 103.17721882113626, 102.52443701939833,
101.88234281789417, 101.25073993042513, 100.62943604419465, 100.01824273369436,
99.41697537653641, 98.82545307118603, 98.24349855655109, 97.670938133386,
97.10760158746663, 96.55332211449726, 96.00793624670783, 95.47128378110379,
94.94320770932941, 94.42355414910807, 93.91217227722247, 93.40891426400009,
92.91363520926834, 92.4261930797463, 91.94644864783902, 91.47426543180346,
91.00950963725302, 90.55205009997074, 90.10175823000112, 89.6585079569906,
89.22217567674824, 88.79264019899907, 88.36978269630175, 87.95348665410482,
87.54363782191439, 87.1401241655488, 86.74283582045447, 86.35166504605935,
85.96650618113907, 85.58725560017427, 85.21381167067483, 84.84607471144975,
84.48394695180062, 84.12733249161803, 83.77613726235991, 83.43026898889143,
83.08963715216751, 82.75415295273805, 82.42372927505734, 82.09828065257932,
81.77772323362082, 81.46197474797505, 81.15095447425865, 80.84458320797523,
80.5427832302789, 80.24547827742292, 79.95259351087628, 79.66405548809483,
79.37979213393038, 79.09973271266458, 78.82380780065256, 78.5519492595631,
78.28409021020138, 78.02016500690165, 77.76010921247575, 77.50385957370658,
77.2513539973727, 77.00253152679286, 76.75733231887865, 76.51569762168332,
76.27756975243618, 76.04289207605115, 75.81160898409905, 75.58366587423325,
75.35900913005794, 75.13758610142965, 74.91934508518213, 74.70423530626476,
74.49220689928568, 74.28321089045018, 74.07719917988547, 73.8741245243434,
73.67394052027281, 73.47660158725252, 73.28206295177768, 73.09028063139093,
72.90121141915142, 72.71481286843319, 72.53104327804624, 72.34986167767323,
72.17122781361377, 71.99510213483074, 71.821445779291, 71.65022056059416,
71.48138895488344, 71.31491408803153, 71.15075972309613, 70.98889024803869,
70.82927066370073, 70.67186657203206, 70.51664416456516, 70.36357021113032,
70.21261204880648, 70.06373757110197, 69.91691521736074, 69.77211396238893,
69.62930330629624, 69.48845326454861, 69.34953435822649, 69.21251760448466,
69.07737450720909, 68.94407704786654, 68.8125976765427, 68.68290930316456,
68.55498528890304, 68.4287994377524, 68.30432598828158, 68.18153960555486,
68.06041537321711, 67.94092878574092, 67.82305574083138, 67.70677253198542,
67.5920558412026, 67.47888273184327, 67.36723064163199, 67.25707737580186,
67.14840110037781, 67.04118033559519, 66.93539394945063, 66.83102115138306,
66.72804148608124, 66.62643482741558, 66.5261813724914, 66.42726163582105,
66.32965644361232, 66.23334692817055, 66.13831452241234, 66.04454095448808,
65.95200824251107, 65.86069868939114, 65.77059487777032, 65.68167966505845,
65.59393617856674, 65.50734781073677, 65.42189821446371, 65.33757129851068,
65.25435122301313, 65.17222239507134, 65.09116946442862, 65.01117731923348,
64.93223108188485, 64.85431610495729, 64.77741796720584, 64.70152246964744,
64.62661563171861, 64.55268368750703, 64.47971308205567, 64.40769046773767,
64.33660270070135, 64.26643683738278, 64.19718013108506, 64.12882002862318,
64.06134416703244, 63.99474037033977, 63.92899664639583, 63.86410118376792,

```
63.8000423486908, 63.73680868207565, 63.67438889657507, 63.612771873703466,
63.551946661011435, 63.49190246931306, 63.432628669965034, 63.374114792196764,
63.316350520490126, 63.259325692007806, 63.20303029406958, 63.14745446167535,
63.092588475073896, 63.038422757376544, 62.98494787221483, 62.932154521441305,
62.88003354287221, 62.8285759080722, 62.77777272017863, 62.72761521176642,
62.67809474275127, 62.62920279833126, 62.580930986965704, 62.53327103839043,
62.48621480166935, 62.43975424328071, 62.39388144523824, 62.348588603245624,
62.30386802488442, 62.259712127834234, 62.21611343812472, 62.17306458841867,
62.13055831632579, 62.088587462746304, 62.047144970243984, 62.00622388144791,
61.96581733748235, 61.925918576424586, 61.88652093178946, 61.84761783104066,
61.809202794128375, 61.77126943205198, 61.73381144544806, 61.69682262320314,
61.66029684109021, 61.62422806042899, 61.588610326769576, 61.553437768598364,
61.5187045960668, 61.484405099741615, 61.45053364937644, 61.41708469270486,
61.38405275425365, 61.35143243417668, 61.31921840710814, 61.28740542103573,
61.25598829619265, 61.22496192396845, 61.19432126583824, 61.16406135230996,
61.13417728188924, 61.10466422006183, 61.07551739829278, 61.04673211304261,
61.01830372479976, 60.99022765712913, 60.96249939573645, 60.93511448754825,
60.90806853980663, 60.88135721917972, 60.854976250886, 60.828921417833605,
60.80318855977331, 60.777773572465684, 60.75267240686181, 60.72788106829721,
60.70339561569895, 60.67921216080579, 60.65532686740061, 60.631735950555715,
60.60843567588992, 60.585422358837796, 60.56269236393055, 60.54024210408869,
60.51806803992567, 60.49616667906292, 60.47453457545572, 60.45316832872977,
60.432064583528245, 60.41122002886948, 60.390631397514426, 60.3702954653444,
60.35020905074846, 60.330369014020526, 60.31077225676591, 60.291415721317136,
60.27229639015889, 60.25341128536208, 60.23475746802646, 60.21633203773212,
60.198132131999465, 60.18015492575748, 60.16239763082013, 60.14485749537116,
60.12753180345645, 60.11041787448444, 60.09351306273412, 60.076814756870476,
60.0603203794675, 60.0440273865383, 60.0279332670726, 60.01203554258093,
59.996331766646115, 59.98081952448119, 59.965496432494284, 59.95036013785987,
59.93540831809657, 59.920638680651344, 59.90604896248978, 59.891636929692595,
59.87740037705821, 59.863337127711134, 59.84944503271633, 59.83572197069925,
59.822165847471325, 59.80877459566148, 59.79554617435247, 59.78247856872311,
59.769569789695524, 59.75681787358758, 59.74422088177056, 59.73177690033175,
59.719484039741964, 59.70734043452786, 59.695344242949396, 59.683493646681406,
59.671786850500226, 59.66022208197484, 59.648797591162314, 59.63751165030773,
59.62636255354846, 59.61534861662274, 59.60446817658236, 59.59371959150957,
59.583101240237866, 59.57261152207704, 59.562248856542006, 59.55201168308545]
```

[8]:
```
plt.plot(epoch, errors, 'r')
plt.xlabel("epoch")
plt.ylabel("error")
plt.show()
```



q2 Consider positive and negative slope dataset given below. Apply logistic regression with gradient descent and illustrate the difference between slope values for both cases at different iterations. Plot the graph of slope(x-axis) vs log-loss (y-axis) for both case separately.

x = np.array([1, 2, 3, 4, 5]) y = np.array([0, 0, 1, 1, 1]) # Positive slope

x = np.array([1, 2, 3, 4, 5]) y = np.array([1, 1, 0, 0, 0]) # Negative slope

[9]:
```
import numpy as np
import pandas as pd

# Create the positive DataFrame
x = np.array([1, 2, 3, 4, 5])
y = np.array([0, 0, 1, 1, 1])
pos_df = pd.DataFrame({"x": x, "y": y})

# Create the negative DataFrame
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 1, 0, 0, 0])
neg_df = pd.DataFrame({"x": x, "y": y})

print(pos_df)
```

```
print(neg_df)
```

```
   x  y
0  1  0
1  2  0
2  3  1
3  4  1
4  5  1
   x  y
0  1  1
1  2  1
2  3  0
3  4  0
4  5  0
```

pos_df :

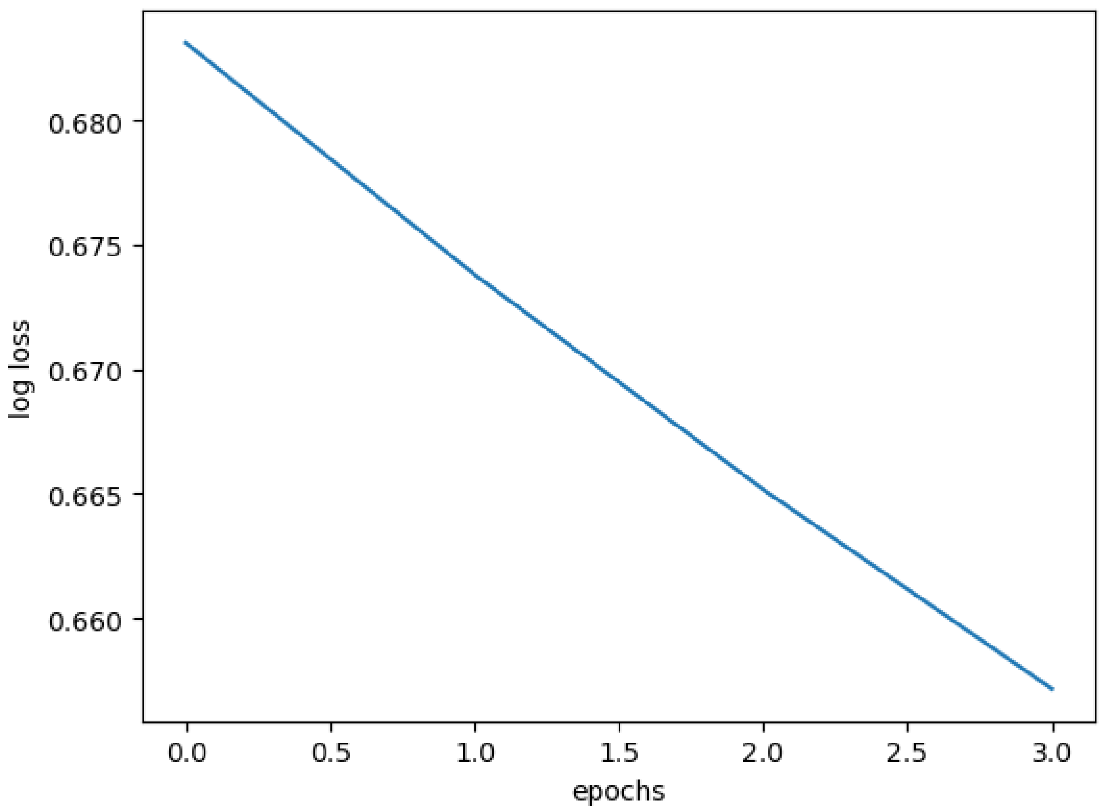
[10]:
```
import numpy as np
import pandas as pd

# Initialize parameters
b0, b1 = 0, 0
alpha = 0.01
epochs = 4
iter_errors = []

# Sample data

df = pos_df

epoch_arr = []
log_loss_arr = []

# Training loop
for j in range(epochs):
    for i in range(df["x"].shape[0]):
        xi = df["x"][i]
        z = b0 + b1 * xi
        pi = 1.0 / (1 + np.exp(-z))  # Predicted probability

        yi = df["y"][i]
        error = yi - pi
        iter_errors.append(abs(error))

        # Update parameters using gradient descent
        b0 += alpha * error * pi * (1 - pi) * 1
        b1 += alpha * error * pi * (1 - pi) * xi
```

```
        # Print the error and updated parameters
        print("Error:", error)
        print("Updated b0, b1:", b0, b1)

    # Calculate and append log loss at the end of each epoch
    y_true = df["y"]
    y_pred = 1.0 / (1 + np.exp(-(b0 + b1 * df["x"])))
    log_loss = -np.mean(y_true * np.log(y_pred + 1e-15) + (1 - y_true) * np.
 ↪log(1 - y_pred + 1e-15))
    epoch_arr.append(j)
    log_loss_arr.append(log_loss)

    print("Epoch", j, "Log Loss:", log_loss)

print("Final parameters: b0 =", b0, ", b1 =", b1)
```

```
Error: -0.5
Updated b0, b1: -0.00125 -0.00125
Error: -0.49906250109863126
Updated b0, b1: -0.002497651866465345 -0.0037453037329306903
Error: 0.503433336802797
Updated b0, b1: -0.0012391278681812901 3.0268261921474054e-05
Error: 0.5002795136760068
Updated b0, b1: 1.1570525150873335e-05 0.005033061835250128
Error: 0.49370611253277286
Updated b0, b1: 0.0012456402345844214 0.011203410382417869
Epoch 0 Log Loss: 0.6831227749911867
Error: -0.5031122224602966
Updated b0, b1: -1.2091590475461593e-05 0.009945678557357986
Error: -0.5049696527210386
Updated b0, b1: -0.0012743910076598143 0.00742107972298928
Error: 0.494752980581478
Updated b0, b1: -3.764476770193033e-05 0.011131318442862932
Error: 0.48887992671218716
Updated b0, b1: 0.0011839505195700584 0.016017699591950887
Error: 0.4796930642825835
Updated b0, b1: 0.0023812050621288532 0.02200397230474486
Epoch 1 Log Loss: 0.6738109674454525
Error: -0.5060959922695626
Updated b0, b1: 0.0011161531524028088 0.020738920395018817
Error: -0.5106468888592617
Updated b0, b1: -0.0001598852196202968 0.018186843650972607
Error: 0.48640319163513146
Updated b0, b1: 0.0010552235303333878 0.02183216990083366
Error: 0.4779183969433515
Updated b0, b1: 0.0022476892060008047 0.02660203260350333
Error: 0.46623699488911896
Updated b0, b1: 0.0034079668688270577 0.032403420917634596
```

```
Epoch 2 Log Loss: 0.6651700686127432
Error: -0.5089518902669914
Updated b0, b1: 0.002135994998573503 0.031131449047381042
Error: -0.5160941614921637
Updated b0, b1: 0.0008470963924382471 0.02855365183511053
Error: 0.47838646434346954
Updated b0, b1: 0.002040827795213665 0.032134846043436784
Error: 0.46740125447116154
Updated b0, b1: 0.003204363960106785 0.03678899070300926
Error: 0.4533487540199761
Updated b0, b1: 0.004327869440543597 0.042406518105193323
Epoch 3 Log Loss: 0.6571575863706118
Final parameters: b0 = 0.004327869440543597 , b1 = 0.042406518105193323
```

[11]:
```python
plt.plot(epoch_arr, log_loss_arr)
plt.xlabel("epochs")
plt.ylabel("log loss")
plt.show()
```



neg_df

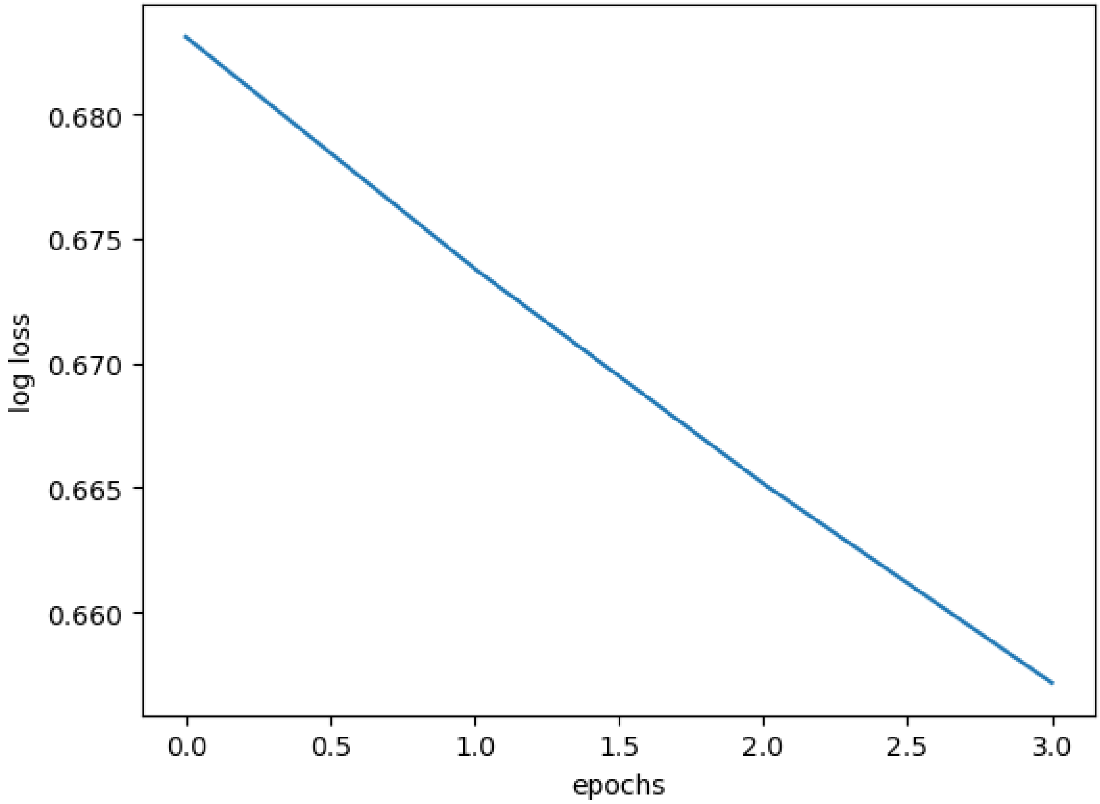
```
[12]: import numpy as np
import pandas as pd

# Initialize parameters
b0, b1 = 0, 0
alpha = 0.01
epochs = 4
iter_errors = []

# Sample data

df = neg_df

epoch_arr = []
log_loss_arr = []

# Training loop
for j in range(epochs):
    for i in range(df["x"].shape[0]):
        xi = df["x"][i]
        z = b0 + b1 * xi
        pi = 1.0 / (1 + np.exp(-z))  # Predicted probability

        yi = df["y"][i]
        error = yi - pi
        iter_errors.append(abs(error))

        # Update parameters using gradient descent
        b0 += alpha * error * pi * (1 - pi) * 1
        b1 += alpha * error * pi * (1 - pi) * xi

        # Print the error and updated parameters
        print("Error:", error)
        print("Updated b0, b1:", b0, b1)

    # Calculate and append log loss at the end of each epoch
    y_true = df["y"]
    y_pred = 1.0 / (1 + np.exp(-(b0 + b1 * df["x"])))
    log_loss = -np.mean(y_true * np.log(y_pred + 1e-15) + (1 - y_true) * np.
 ↪log(1 - y_pred + 1e-15))
    epoch_arr.append(j)
    log_loss_arr.append(log_loss)

    print("Epoch", j, "Log Loss:", log_loss)

print("Final parameters: b0 =", b0, ", b1 =", b1)
```

```
Error: 0.5
```

```
Updated b0, b1: 0.00125 0.00125
Error: 0.49906250109863126
Updated b0, b1: 0.002497651866465345 0.0037453037329306903
Error: -0.5034333368027969
Updated b0, b1: 0.0012391278681812903 -3.0268261921473187e-05
Error: -0.5002795136760068
Updated b0, b1: -1.1570525150873118e-05 -0.005033061835250127
Error: -0.493706112532773
Updated b0, b1: -0.0012456402345844217 -0.011203410382417869
Epoch 0 Log Loss: 0.6831227749911867
Error: 0.5031122224602966
Updated b0, b1: 1.2091590475461376e-05 -0.009945678557357986
Error: 0.5049696527210387
Updated b0, b1: 0.0012743910076598146 -0.007421079722989279
Error: -0.4947529805814781
Updated b0, b1: 3.764476770193033e-05 -0.011131318442862932
Error: -0.48887992671218716
Updated b0, b1: -0.0011839505195700587 -0.016017699591950887
Error: -0.47969306428258346
Updated b0, b1: -0.0023812050621288532 -0.02200397230474486
Epoch 1 Log Loss: 0.6738109674454525
Error: 0.5060959922695626
Updated b0, b1: -0.0011161531524028088 -0.020738920395018817
Error: 0.5106468888592617
Updated b0, b1: 0.00015988521962029658 -0.018186843650972607
Error: -0.4864031916351315
Updated b0, b1: -0.0010552235303333884 -0.021832169900833663
Error: -0.4779183969433515
Updated b0, b1: -0.0022476892060008056 -0.026602032603503332
Error: -0.46623699488911896
Updated b0, b1: -0.0034079668688270586 -0.032403420917634596
Epoch 2 Log Loss: 0.6651700686127434
Error: 0.5089518902669914
Updated b0, b1: -0.0021359949985735043 -0.031131449047381042
Error: 0.5160941614921637
Updated b0, b1: -0.0008470963924382484 -0.02855365183511053
Error: -0.4783864643434696
Updated b0, b1: -0.00204082779521 3667 -0.03213484604343 6784
Error: -0.46740125447116143
Updated b0, b1: -0.0032043639601067866 -0.03678899070300926
Error: -0.45334875401997615
Updated b0, b1: -0.0043278694405436 -0.042406518105193323
Epoch 3 Log Loss: 0.6571575863706117
Final parameters: b0 = -0.0043278694405436 , b1 = -0.042406518105193323
```

[13]:
```
plt.plot(epoch_arr, log_loss_arr)
plt.xlabel("epochs")
```

```
plt.ylabel("log loss")
plt.show()
```



Create the following data set for Experience and Salary in CSV. Applying SLR, explore the relationship between salary and experience with exerience in x-axis and salary in y axis.

a. Check for various values of beta (slope) = 0.1, 1.5, and 0.8 with a fixed value of intercept i.e b=1.1. Plot the graph between beta and mean squared error(MSE) for each case.

b. Try with beta between 0 to 1.5 with an increment of 0.01 keeping b (intercept) as constant and Plot the graph between beta and mean squared error(MSE).

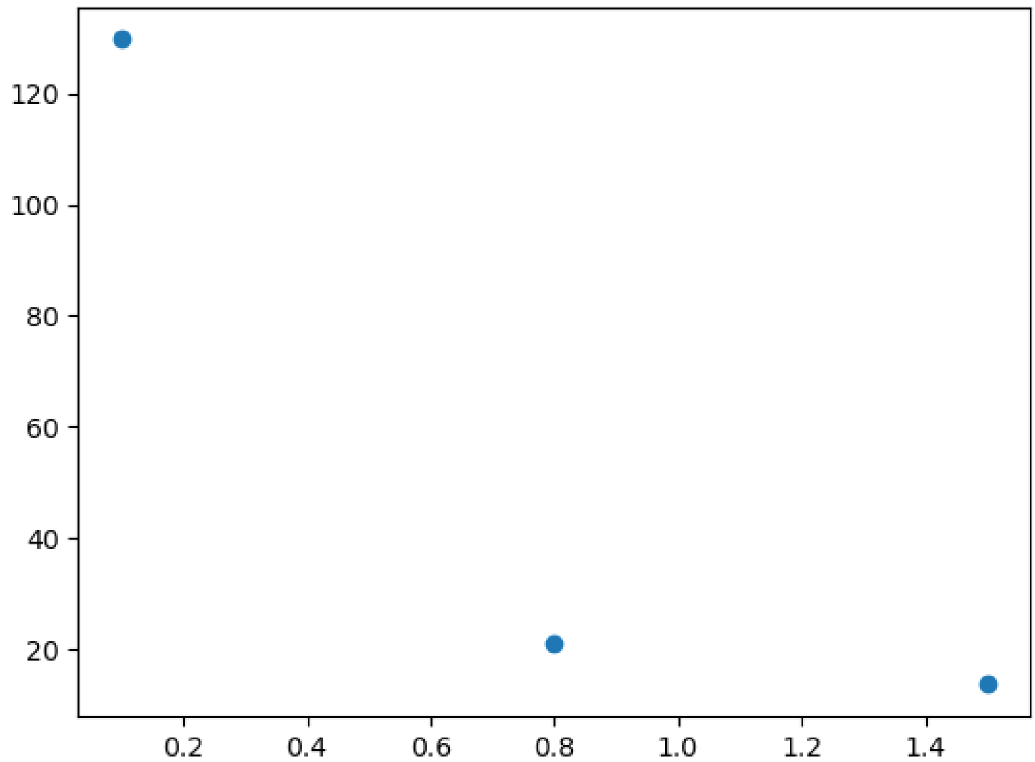
c. Try with different values of intercept for slope beta between 0 to 1.5 with an increment of 0.01. Plot the graph between beta and mean squared error(MSE).

d. Use the scikit learn and compare the results of MSE.

[14]:
```
#### a
```

[23]:
```
df = sal_df
b0 = 1.1 #given
b1_arr = [0.1,1.5,0.8]
```

```
mse_arr = []

for b1 in b1_arr:
    predictions = b0 + b1*df["experience"]
    mse = np.sum((predictions - df["salary"])**2)
    mse_arr.append(mse)

plt.scatter(b1_arr, mse_arr)
plt.show()
```



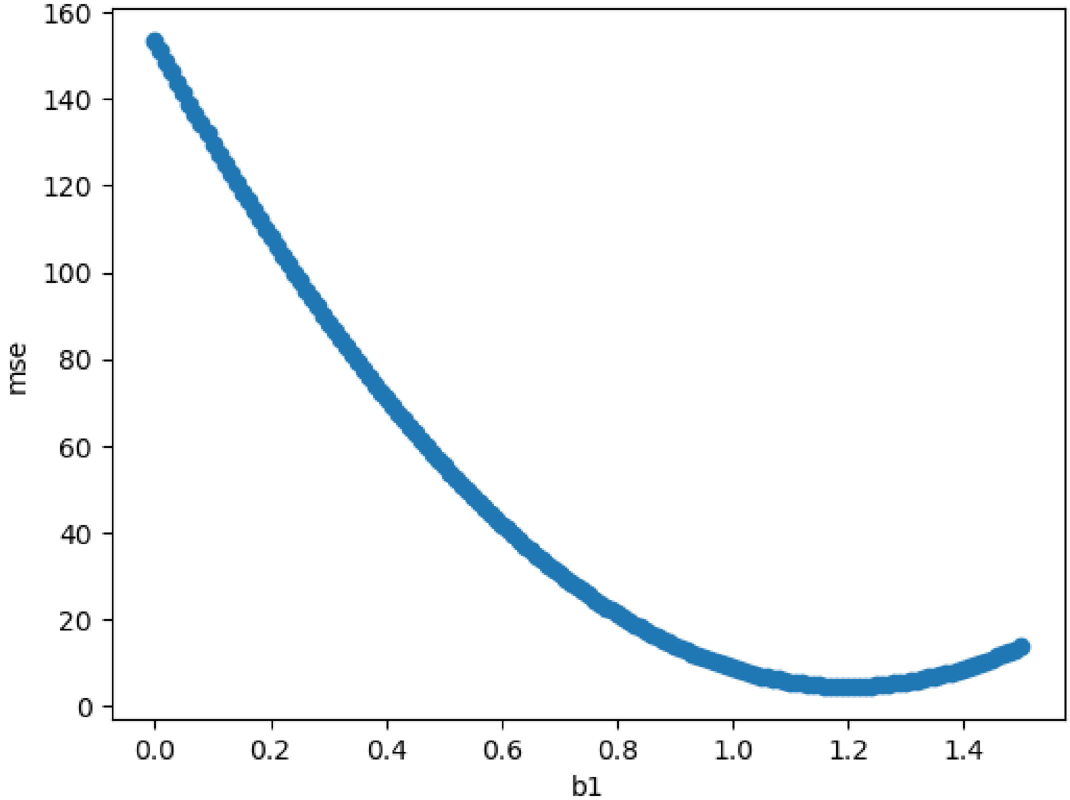
[27]:

```
#### b
```

[28]:

```
import numpy as np

# Generate values from 0 to 1.5 with a step size of 0.01
b1_arr = np.arange(0, 1.51, 0.01)  # Note: 1.51 is used to include 1.5 in the
 ↪range

mse_arr = []

for b1 in b1_arr:
```

```
    predictions = b0 + b1*df["experience"]
    mse = np.sum((predictions - df["salary"])**2)
    mse_arr.append(mse)

plt.scatter(b1_arr, mse_arr)
plt.xlabel("b1")
plt.ylabel("mse")
plt.show()
```



c (same as b) Try with different values of intercept for slope beta between 0 to 1.5 with an increment of 0.01. Plot the graph between beta and mean squared error(MSE)

d Use the scikit learn and compare the results of MSE

[ ]:

```

```

[32]:

```
data = {
    "time": [1,2,3,4,5,6,7,8],
    "pass": [0,0,0,0,1,1,1,1]
}
```

```
df = pd.DataFrame(data)
```

4 Apply Stochastic Gradient Descent for the afore-mentioned dataset, and arrive at different values of B0, B1 and error for 60 iterations of 5 epochs.

a. Plot the graph of log loss/error versus iteration.

b.Use the scikit learn and arrive at the results of B0, B1 and error, for 60 iterations of 5 epochs.

c. Plot the graph between beta (X-axis) and log loss/ error (Y-axis) using scikit learn and your approach separately.
d. Plot the separate graph of –log(x) (y=1 case) and –log(1-x) (y=0 case) and also draw the combined graph of .bothcases

[33]:
```
#### a
```

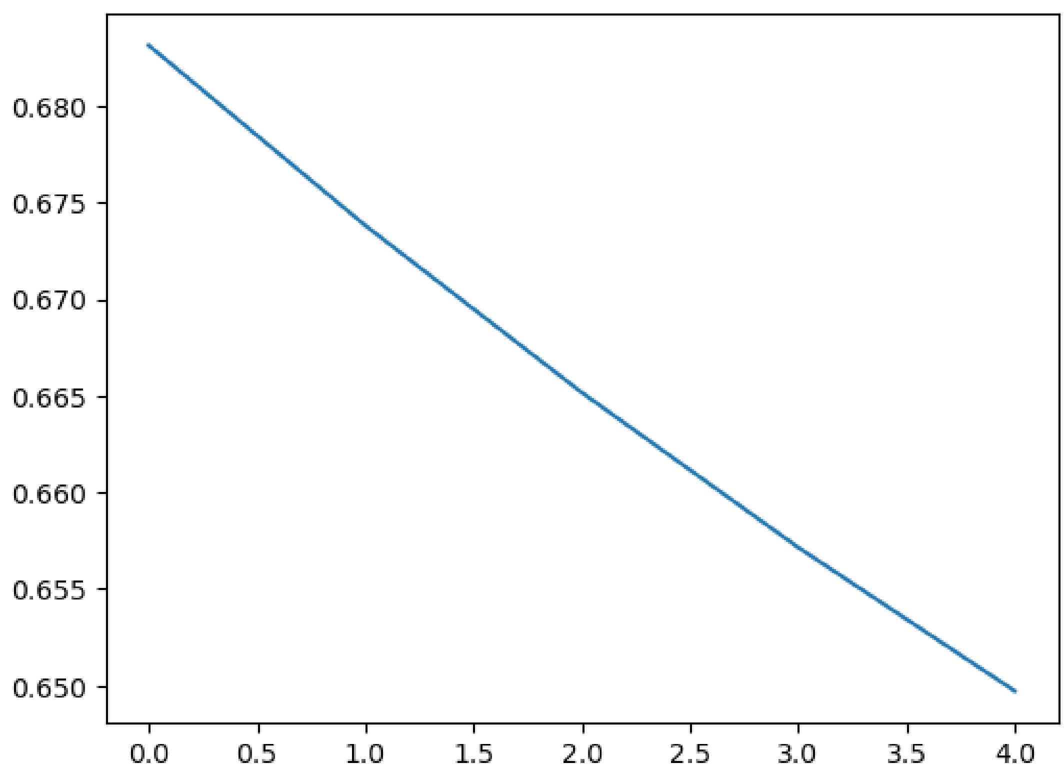
[39]:
```
import numpy as np
import pandas as pd

# Initialize parameters
b0, b1 = 0, 0
alpha = 0.01
epochs = 5
iter_errors = []
log_loss_arr = []
epoch_arr = []

# Sample DataFrame (make sure your actual DataFrame is defined similarly)
x = np.array([1, 2, 3, 4, 5])
y = np.array([0, 0, 1, 1, 1])
df = pd.DataFrame({"time": x, "pass": y})

# Training loop
for epoch in range(epochs):
    for i in range(df["pass"].shape[0]):
        xi = df["time"][i]
        z = b0 + b1 * xi
        pi = 1.0 / (1 + np.exp(-z))  # Predicted probability

        yi = df["pass"][i]
        error = yi - pi
        iter_errors.append(abs(error))

        # Update parameters using gradient descent
        b0 += alpha * error * pi * (1 - pi) * 1
        b1 += alpha * error * pi * (1 - pi) * xi

    # Calculate log loss at the end of each epoch
```

```
    y_true = df["pass"].values
    y_pred = 1.0 / (1 + np.exp(-(b0 + b1 * df["time"].values)))
    log_loss = -np.mean(y_true * np.log(y_pred + 1e-15) + (1 - y_true) * np.
↪log(1 - y_pred + 1e-15))
    epoch_arr.append(epoch)
    log_loss_arr.append(log_loss)

    print(f"Epoch {epoch}: Log Loss = {log_loss}")

print("Final parameters: b0 =", b0, ", b1 =", b1)
```

```
Epoch 0: Log Loss = 0.6831227749911867
Epoch 1: Log Loss = 0.6738109674454525
Epoch 2: Log Loss = 0.6651700686127432
Epoch 3: Log Loss = 0.6571575863706118
Epoch 4: Log Loss = 0.6497311445496404
Final parameters: b0 = 0.005143401572145189 , b1 = 0.0520205218124939
```

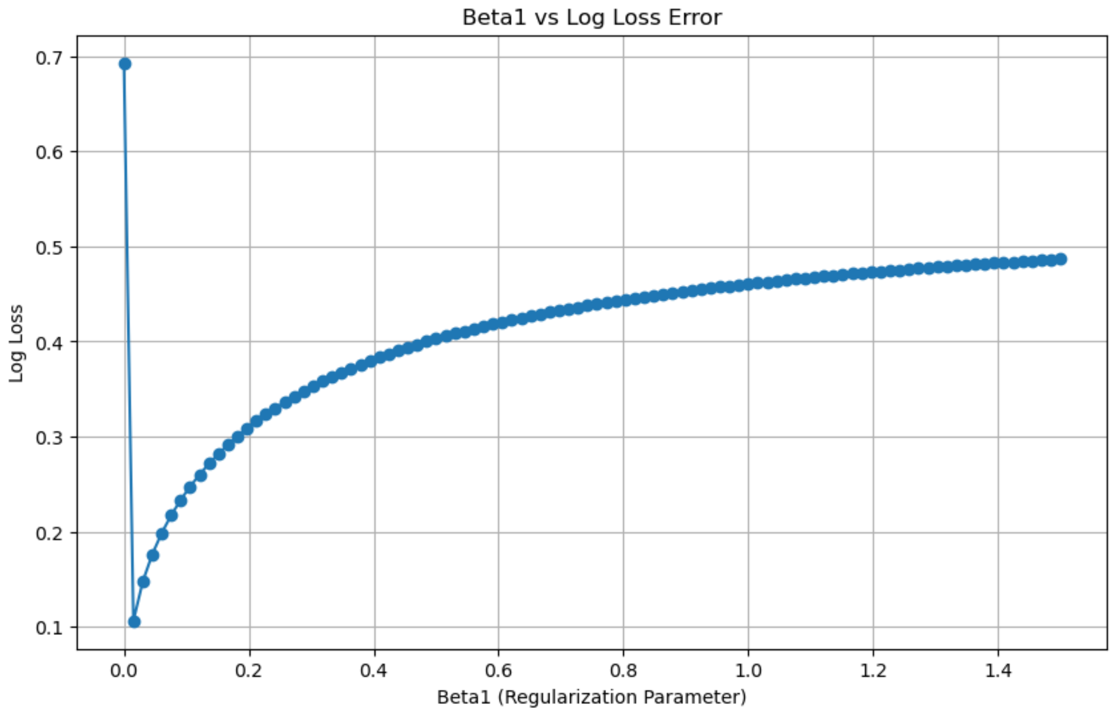
[41]:
```
plt.plot(log_loss_arr)
```

[41]: [<matplotlib.lines.Line2D at 0x7095ceafffe0>]



[42]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
from sklearn.model_selection import train_test_split

# Generate sample data
x = np.array([1, 2, 3, 4, 5])
y = np.array([0, 0, 1, 1, 1])
df = pd.DataFrame({"time": x, "pass": y})

# Prepare data for scikit-learn
X = df[['time']]
y = df['pass']

# Define range of beta1 values
beta1_values = np.linspace(0, 1.5, 100)
log_loss_arr = []

for beta1 in beta1_values:
    # Train a logistic regression model with the current beta1
    model = LogisticRegression(solver='liblinear', C=1/beta1 if beta1 != 0 else␣
↪1e-10)  # Note: C is the inverse of regularization strength
    model.fit(X, y)

    # Predict probabilities
    y_pred = model.predict_proba(X)[:, 1]

    # Calculate log loss
    loss = log_loss(y, y_pred)
    log_loss_arr.append(loss)

# Plot beta1 vs. log loss
plt.figure(figsize=(10, 6))
plt.plot(beta1_values, log_loss_arr, marker='o')
plt.xlabel('Beta1 (Regularization Parameter)')
plt.ylabel('Log Loss')
plt.title('Beta1 vs Log Loss Error')
plt.grid(True)
plt.show()
```



d Plot the separate graph of –log(x) (y=1 case) and –log(1-x) (y=0 case) and also draw the combined graph of .both cases.

[ ]:

[ ]:

[ ]:

[ ]:

Use hours vs Fail/ Pass Data from previos lab for this quesitrons

5 Consider positive and negative slope dataset given below. Apply simple linear regression with gradient descent and illustrate the difference between slope values for both cases at different iterations. Plot the graph of slope(x- axis) vs MSE (y-axis) for both case separately.

x = np.array([1, 2, 4, 3, 5])

y = np.array([1, 3, 3, 2, 5]) # Positive slope

x = np.array([1, 2, 3, 4, 5])

y = np.array([10, 8, 6, 4, 2]) # Negative slope

[43]:
```
x = np.array([1, 2, 3, 4, 5])
y = np.array([0, 0, 1, 1, 1])
pos_df = pd.DataFrame({"x": x, "y": y})
x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 8, 6, 4, 2])
neg_df = pd.DataFrame({"x": x, "y": y})
```

[53]:
```
#performing gradient descent
df = pos_df

b0, b1 = 0, 0
alpha = 0.0001

epoch_error = []
slope_arr = []
mse_arr = []

#stochastic (example by example)
epocs = 4
for _ in range(epocs):
    for i in range(df.shape[0]):
        xi = df["x"][i]
        #pi = b0 + b1*xi
        pi = b0 + b1*xi
        yi = df["y"][i]
        error = pi-yi
        epoch_error.append(abs(error))
        b0 = b0 - alpha*error
        b1 = b1 - alpha*error*xi
        slope_arr.append(b1)
        print(error)
        print("updated b0, b1: ", b0, b1)
        if i == df.shape[0]-1:
            prediction = b0 + b1*df["x"]
            mse = np.sum(((prediction - df["y"])**2) / df.shape[0])
            # print(mse)
            mse_arr.append(mse)
            rmse = mse**0.5

            print("mse: ", mse)
            print("rmse: ", rmse)

        # print(b0, b1)

prediction = b0 + b1*df["x"]
print(prediction)
```

```
0
```

```
updated b0, b1:  0.0 0.0
0.0
updated b0, b1:  0.0 0.0
-1.0
updated b0, b1:  0.0001 0.00030000000000000003
-0.9987
updated b0, b1:  0.00019987 0.00069948
-0.99630273
updated b0, b1:  0.000299500273 0.001197631365
mse:  0.5939099884960888
rmse:  0.7706555576235655
0.0014971316380000001
updated b0, b1:  0.0002993505598362 0.0011974816518362
0.0026943138635086
updated b0, b1:  0.00029908112844984915 0.0011969427890634982
-0.9961100905043596
updated b0, b1:  0.0003986921375002851 0.0014957758162148062
-0.9936182045976405
updated b0, b1:  0.0004980539579600492 0.0018932230980538624
-0.9900358305517707
updated b0, b1:  0.0005970575410152263 0.002388241013329748
mse:  0.5878916267148621
rmse:  0.7667409123784005
0.0029852985543449744
updated b0, b1:  0.0005967590111597918 0.0023879424834743136
0.005372643978108419
updated b0, b1:  0.000596221746761981 0.002386867954678692
-0.992243174389202
updated b0, b1:  0.0006954460642009012 0.0026845409069954527
-0.9885663903078172
updated b0, b1:  0.0007943027032316829 0.00307996746311858
-0.9838058599811754
updated b0, b1:  0.0008926832892298005 0.0035718703931091676
mse:  0.5819440711826793
rmse:  0.7628525881077414
0.004464553682338968
updated b0, b1:  0.0008922368338615666 0.0035714239377409337
0.008035084709343434
updated b0, b1:  0.0008914333253906323 0.003569816920799065
-0.9883991159122122
updated b0, b1:  0.0009902732369818535 0.0038663366555727286
-0.9835443801407272
updated b0, b1:  0.0010886276749959261 0.004259754407629019
-0.977612600286859
updated b0, b1:  0.001186388935024612 0.004748560707772449
mse:  0.5760664883554032
rmse:  0.7589904402266231
0    0.005935
```

```
1    0.010684
2    0.015432
3    0.020181
4    0.024929
Name: x, dtype: float64
```

[56]:

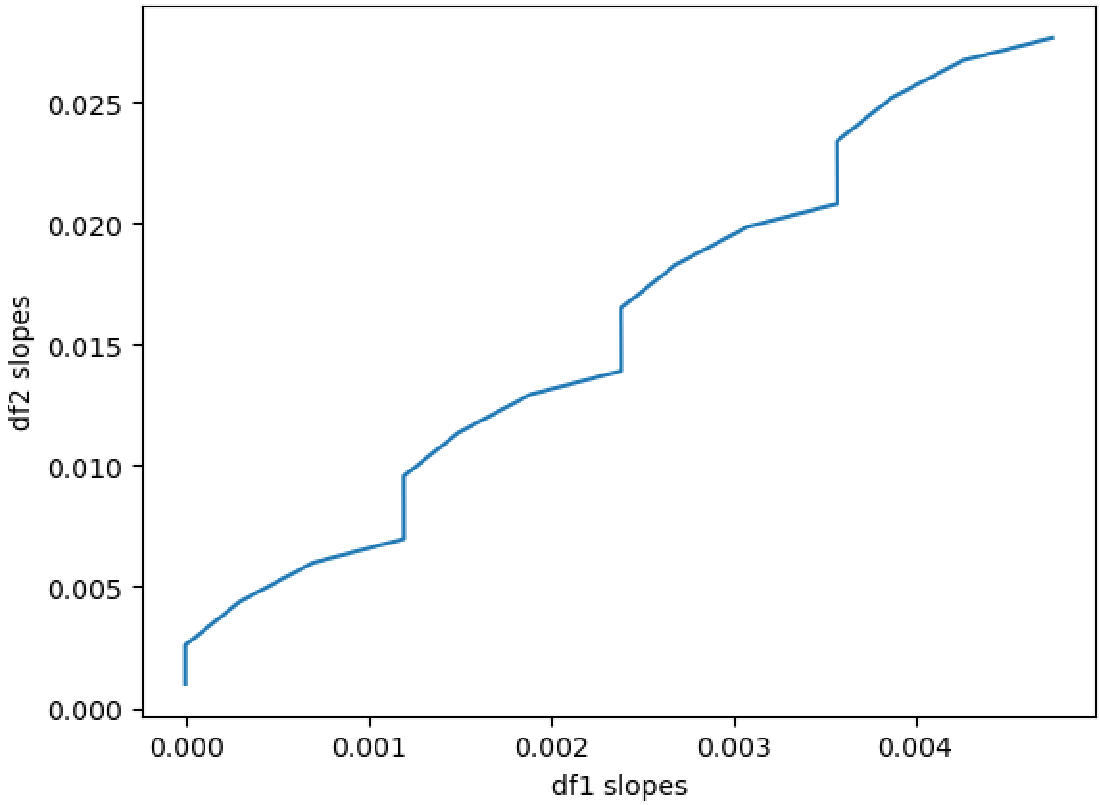
```
#performing gradient descent
df = neg_df

b0, b1 = 0, 0
alpha = 0.0001

epoch_error = []
neg_slope_arr = []
neg_mse_err = []

#stochastic (example by example)
epocs = 4
for _ in range(epocs):
    for i in range(df.shape[0]):
        xi = df["x"][i]
        #pi = b0 + b1*xi
        pi = b0 + b1*xi
        yi = df["y"][i]
        error = pi-yi
        epoch_error.append(abs(error))
        b0 = b0 - alpha*error
        b1 = b1 - alpha*error*xi
        neg_slope_arr.append(b1)
        print(error)
        print("updated b0, b1: ", b0, b1)
        if i == df.shape[0]-1:
            prediction = b0 + b1*df["x"]
            mse = np.sum(((prediction - df["y"])**2) / df.shape[0])
            # print(mse)
            mse_arr.append(mse)
            rmse = mse**0.5

            print("mse: ", mse)
            print("rmse: ", rmse)

        # print(b0, b1)

prediction = b0 + b1*df["x"]
print(prediction)
```

```
-10
updated b0, b1:  0.001 0.001
```

```
-7.997
updated b0, b1:  0.0017997 0.0025994
-5.9904021
updated b0, b1:  0.00239874021 0.00439652063
-3.98001517727
updated b0, b1:  0.002796741727727 0.005988526700908
-1.9672606247677331
updated b0, b1:  0.0029934677902O3773 0.006972157013291866
mse:  43.76952689726708
rmse:  6.615854207679238
-9.990034375196505
updated b0, b1:  0.0039924712277234235 0.007971160450811516
-7.980065207870654
updated b0, b1:  0.004790477748510489 0.009567173492385648
-5.966508001774333
updated b0, b1:  0.005387128548687923 0.011357125892917947
-3.9491843678796403
updated b0, b1:  0.005782046985475887 0.012936799640069803
-1.929533954814175
updated b0, b1:  0.005975000380957305 0.013901566617476891
mse:  43.541715991062354
rmse:  6.598614702425226
-9.980123433001566
updated b0, b1:  0.006973012724257462 0.014899578960777048
-7.963227829354189
updated b0, b1:  0.007769335507192881 0.016492224526647885
-5.9427539909128635
updated b0, b1:  0.008363610906284168 0.018275050723921743
-3.918536186198029
updated b0, b1:  0.00875546452490397 0.019842465198400955
-1.8920322094830913
updated b0, b1:  0.00894466774585228 0.0207884813031425
mse:  43.31653596449585
rmse:  6.581529910628368
-9.970266850951004
updated b0, b1:  0.00994169443094738 0.021785507988237602
-7.946487289592578
updated b0, b1:  0.010736343159906637 0.023374805446156117
-5.919139240501625
updated b0, b1:  0.0113282570839568 0.025150547218306604
-3.8880695540428167
updated b0, b1:  0.011717064039361081 0.026705775039923732
-1.8547540607610202
updated b0, b1:  0.011902539445437184 0.027633152070304243
mse:  43.09395586925335
rmse:  6.564598683031077
0    0.039536
1    0.067169
```

```
2    0.094802
3    0.122435
4    0.150068
Name: x, dtype: float64
```

[57]:
```
plt.plot(slope_arr, neg_slope_arr)
plt.xlabel("df1 slopes")
plt.ylabel("df2 slopes")
plt.show()
```

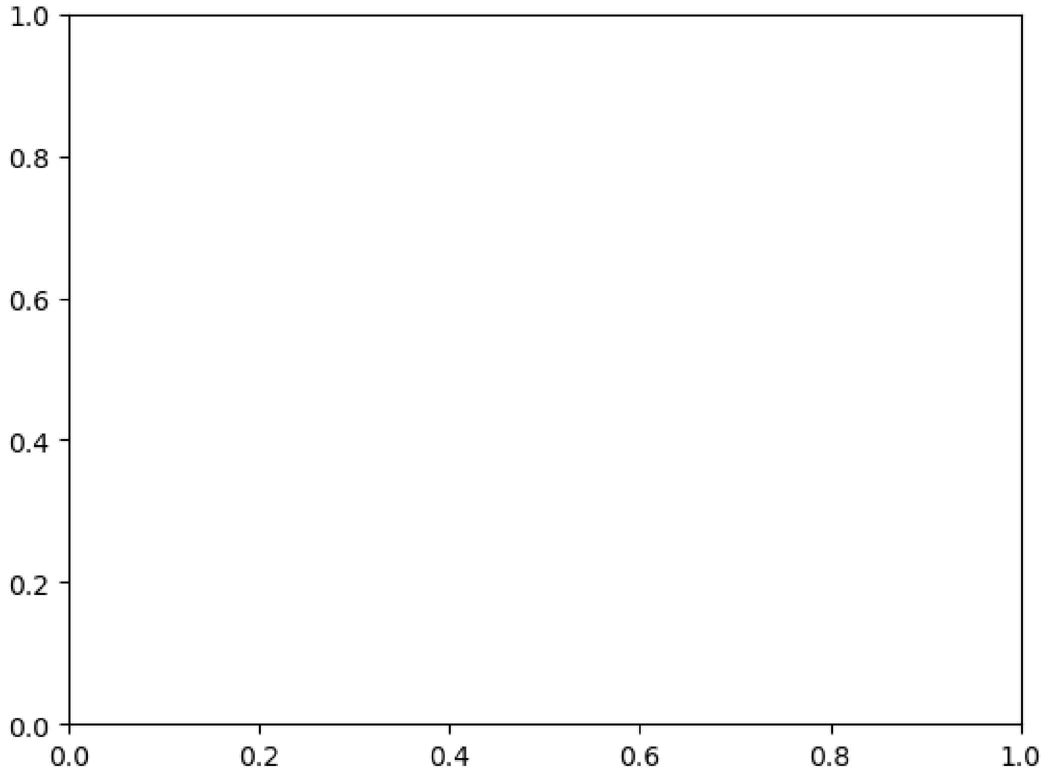


Plot the graph of slope(x- axis) vs MSE (y-axis) for both case separately.

[59]:
```
plt.plot(slope_arr, mse_arr)
plt.plot(neg_slope_arr, neg_mse_arr)
plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[59], line 1
----> 1 plt.plot(slope_arr, mse_arr)
      2 plt.plot(neg_slope_arr, neg_mse_arr)
      3 plt.show()
```

```
File /usr/lib/python3/dist-packages/matplotlib/pyplot.py:2748, in plot(scalex, scaley, data, *args, **kwargs)
   2746 @_copy_docstring_and_deprecators(Axes.plot)
   2747 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2748     return gca().plot(
   2749         *args, scalex=scalex, scaley=scaley,
   2750         **({"data": data} if data is not None else {}), **kwargs)

File /usr/lib/python3/dist-packages/matplotlib/axes/_axes.py:1668, in Axes.plot(self, scalex, scaley, data, *args, **kwargs)
   1425 """
   1426 Plot y versus x as lines and/or markers.
   1427
   (…)
   1665 (``'green'``) or hex strings (``'#008000'``).
   1666 """
   1667 kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1668 lines = [*self._get_lines(*args, data=data, **kwargs)]
   1669 for line in lines:
   1670     self.add_line(line)

File /usr/lib/python3/dist-packages/matplotlib/axes/_base.py:311, in _process_plot_var_args.__call__(self, data, *args, **kwargs)
    309     this += args[0],
    310     args = args[1:]
--> 311 yield from self._plot_args(
    312     this, kwargs, ambiguous_fmt_datakey=ambiguous_fmt_datakey)

File /usr/lib/python3/dist-packages/matplotlib/axes/_base.py:504, in _process_plot_var_args._plot_args(self, tup, kwargs, return_kwargs, ambiguous_fmt_datakey)
    501     self.axes.yaxis.update_units(y)
    503 if x.shape[0] != y.shape[0]:
--> 504     raise ValueError(f"x and y must have same first dimension, but "
    505                      f"have shapes {x.shape} and {y.shape}")
    506 if x.ndim > 2 or y.ndim > 2:
    507     raise ValueError(f"x and y can be no greater than 2D, but have "
    508                      f"shapes {x.shape} and {y.shape}")

ValueError: x and y must have same first dimension, but have shapes (20,) and (8,)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: