

```
1 #Wesley Johanson
2 # from aifc import _Marker
3 # from re import I
4 from re import I
5 import matplotlib as mpl
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import matplotlib.font_manager as fm
9 from sklearn.linear_model import LinearRegression
10 from scipy import stats
11 import random
12
13 class ChEplot:
14     def __init__(self):
15         self.figure=None
16         self.dataLabels=None
17         self.dataColors =None
18         #Counting Elements
19         self.numDataVars=None
20         self.numDataFns=None
21         self.numDataSets=None
22         self.data=None
23         self.fxns2plot=None
24
25     #Data setter/getter/modifier functions
26     # def loadCSV(self, filename: str,folder=None, names=None, indepVars=1, skip=0):
27     #     """Loads each column of data from the CSV file into a row of a numpy
28     #     array stored in self.data
29
30     #     'names' is a list of names for the data sets in each col of the CSV"""
31     #     self.data = np.loadtxt(filename, unpack=True, \
32     #                             delimiter=',',skiprows=skip)
33     #     if names is not None: self.dataLabels = names
34     #     self.numDataVars = indepVars
35     #     self.numDataSets = len(self.data)
36     #     self.numDataFns = self.numDataSets - self.numDataVars
37
38
39     #Data
40     def loadLabels(self):
41         pass
42
43     def loadCSV(self, filename: str, names: list, indepVars, skip=0):
44         """Loads each column of data from the CSV file into a row of a numpy
45         array stored in self.data
46
47         'names' is a list of names for the data sets in each col of the CSV"""
48         # if indepVars < 1 or indepVars > len(names): return
49         self.data = np.loadtxt(filename, unpack=True, delimiter=',',skiprows=skip)
50         # if indepVars > self.numDataSets: self.data = none; return
51         self.dataLabels = names
52         self.numDataVars = indepVars
53         self.numDataSets = len(self.data)
54         self.numDataFns = self.numDataSets - self.numDataVars
55
56     def loadCSV_str(self, filename: str, names: list, indepVars, skip=0):
57         """Loads each column of data from the CSV file into a row of a numpy
```

```

58     array stored in self.data
59
60     'names' is a list of names for the data sets in each col of the CSV"""
61     # if indepVars < 1 or indepVars > len(names): return
62     self.data = np.loadtxt(filename, unpack=True, delimiter=',', skiprows=skip,
dtype=str)
63     # if indepVars > self.numDataSets: self.data = None; return
64     self.dataLabels = names
65     self.numDataVars = indepVars
66     self.numDataSets = len(self.data)
67     self.numDataFns = self.numDataSets - self.numDataVars
68     # for col in range(0, len(self.data)):
69     #     new_x = []
70     #     new_y = []
71     #     for row in range(0, len(self.data[0])):
72     #         if self.data[col,row] != "":
73     #             new_x.append(float(self.data[col,row]))
74     #             new_y.append(float(self.data[0,row]))
75
76
77
78
79     def setData(self, data: list, vars=1):
80         "Replaces Data and performs same operations as loadCSV"
81         self.data = data
82         self.numDataVars = vars
83         self.numDataSets = len(self.data)
84         self.numDataFns = self.numDataSets - self.numDataVars
85
86
87     #Printers
88     def printAllData(self):
89         print(
90             "\n",self.figure
91             ,"\n",self.dataLabels
92             ,"\n",self.numDataVars
93             ,"\n",self.numDataFns
94             ,"\n",self.numDataSets
95             ,"\n",self.data
96             ,"\n",self.fxns2plot)
97         pass
98
99     def printData(self):
100         "print all data points in self.data"
101         print(self.data)
102
103     def printMeans(self):
104         "Prints the mean value of each row vector in self.data"
105         for i in range(0, self.numDataSets):
106             outputStr = "the mean of "
107             if self.dataLabels[i] is not None:
108                 outputStr += self.dataLabels[i]
109             outputStr += "\t\tis " + str(np.mean(self.data[i]))
110
111     #Setters
112     def setDataLabel(self, names):
113         """
114         Stores a list of strings into the instance, where each str in the list

```

```
115         is the name of the corresponding column in the CSV file
116         """
117         self.dataLabels = names
118
119     def segmentData(self):
120         pass
121
122     def setLRegLineColors(self, colors=[]):
123         self.LRegLineColors = colors
124
125     def setIndepVars(self, vars):
126         "Vars are the first arrays in the self.data matrix"
127         self.setIndepVars = vars
128
129     #Plotters
130     def plotData(self, width, height, markers=None):
131         self.figure = plt.figure(figsize=(width, height))
132         L, B, W, H = [0.15, 0.1, 0.80, 0.85]
133         self.figure.axis = []
134         self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
135         #find a way to exclude data
136         # for var in range(0, self.numDataVars):
137         var = self.fxns2plot[0]
138         for fn in self.fxns2plot[1:]:
139             x = self.data[var]
140             y = self.data[fn]
141             #LOOK
142             lbl = self.dataLabels[fn]
143             # random_color=list(np.random.choice(range(255),size=3))
144             if markers is not None:
145                 mk = markers[fn]
146             else:
147                 mk = "."
148             if self.dataColors is not None:
149                 clr = self.dataColors[fn]
150                 self.figure.axis[0].plot(x,y,mk,label=lbl,color=clr)
151             else:
152                 self.figure.axis[0].plot(x,y,mk,label=lbl)
153
154     def plotData_str(self, width, height, markers=None):
155         self.figure = plt.figure(figsize=(width, height))
156         L, B, W, H = [0.15, 0.1, 0.80, 0.85]
157         self.figure.axis = []
158         self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
159         #find a way to exclude data
160         # for var in range(0, self.numDataVars):
161         var = self.fxns2plot[0]
162         for fn in self.fxns2plot[1:]:
163             x = self.data[var]
164             y = self.data[fn]
165             #LOOK
166             lbl = self.dataLabels[fn]
167             # random_color=list(np.random.choice(range(255),size=3))
168             for col in range(0, len(self.data)):
169                 new_x = []
170                 new_y = []
171                 for row in range(0, len(self.data[0])):
```

```

172         if self.data[col,row] != "":
173             new_x.append(float(self.data[var,row]))
174             new_y.append(float(self.data[col,row]))
175     if markers is not None:
176         mk = markers[fn]
177     else:
178         mk = "."
179     if self.dataColors is not None:
180         clr = self.dataColors[fn]
181         self.figure.axis[0].plot(new_x,new_y,mk,label=lbl,color=clr)
182     else:
183         self.figure.axis[0].plot(new_x,new_y,mk,label=lbl)
184
185 def plotData_str_logx(self, width, height, markers=None):
186     self.figure = plt.figure(figsize=(width, height))
187     L, B, W, H = [0.15, 0.1, 0.80, 0.85]
188     self.figure.axis = []
189     self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
190     #find a way to exclude data
191     # for var in range(0, self.numDataVars):
192     var = self.fxns2plot[0]
193     for fn in self.fxns2plot[1:]:
194         x = self.data[var]
195         y = self.data[fn]
196         #LOOK
197         lbl = self.dataLabels[fn]
198         # random_color=list(np.random.choice(range(255),size=3))
199         for col in range(0, len(self.data)):
200             new_x = []
201             new_y = []
202             for row in range(0, len(self.data[0])):
203                 if self.data[col,row] != "":
204                     new_x.append(float(self.data[var,row]))
205                     new_y.append(float(self.data[col,row]))
206             if markers is not None:
207                 mk = markers[fn]
208             else:
209                 mk = "."
210             if self.dataColors is not None:
211                 clr = self.dataColors[fn]
212                 self.figure.axis[0].plot(new_x,new_y,mk,label=lbl,color=clr)
213             else:
214                 self.figure.axis[0].plot(new_x,new_y,mk,label=lbl)
215     self.figure.axis[0].set_xscale('log')
216     # self.figure.axis[0].set_yscale('log')
217
218 def plotLRegLines(self, width=0.5, style='-'):
219     var = self.fxns2plot[0]
220     fxns = self.fxns2plot[1:]
221     for fn in fxns:
222         m, b = np.polyfit(self.data[var],self.data[fn],1)
223         x = np.linspace(min(self.data[var]), max(self.data[var]),
num=len(self.data[var]))
224         y = m * x + b
225         txt1 = "LinReg line for y = " + self.dataLabels[fn]
226         txt2 = "and x = " + self.dataLabels[var]
227         txt3 = "y = (%.4f" % m + ")x + (%.4f" % b + ")"
228         txt4 = "R^2 = %.4f" % ChEplot.rSquared(self.data[var], self.data[fn])

```

```

229     print( f"{txt1:<35}{txt2:<30}{txt3:>20}{txt4:>20}" )
230     if self.dataColors is None:
231         self.figure.axis[0].plot(x, y, \
232             linewidth=width ,linestyle=style)
233     else:
234         self.figure.axis[0].plot(x, y, \
235             color=self.dataColors[fn], \
236             linewidth=width ,linestyle=style)
237
238     def plotErrorBars(self):
239         # stdDev =
240         pass
241
242     #Statistics
243     @staticmethod
244     def rSquared(x, y):
245         x = x.reshape((-1,1))
246         y_reg = LinearRegression().fit(x,y)
247         return y_reg.score(x,y)
248
249     def printAllRSquared(self, precision=5, vars=None, fxns=None):
250         if vars is None:
251             vars = range(0, self.numDataSets)
252         if fxns is None:
253             fxns = range(0, self.numDataSets)
254
255         for fn in fxns:
256             for var in vars:
257                 if fn == var:
258                     continue
259                 rSquared = ChEplot.rSquared(self.data[0],self.data[fn])
260                 rStr = "R^2 = %1." + str(precision) + "f"
261                 rStr = rStr % rSquared
262                 if rStr is None: print("Error_0")
263                 if self.dataLabels is None: print("Error_1")
264                 print( f"{rStr:<25}{self.dataLabels[fn-self.numDataVars]:<20}{'with
respect to':<20}{self.dataLabels[var]:>10}" )
265
266     #COMPLETE ME
267     def confInterv(self, n=1):
268         self.lowerBound_CI = []
269         self.upperBound_CI = []
270         for fn in range(self.numDataVars, self.numDataSets):
271             x_bar , stdDev = np.mean(self.data[fn]) ,np.std(self.data[fn])
272             SE = stdDev / math.sqrt(self.num)
273             DoF = n
274             stats.t.ppf(q=0.05,)
275             scipy.stats.t.ppf(q=.05,df=22)
276
277
278     #Plot: Setters
279     def setFxns2Plot(self, fxns):
280         self.fxns2plot = fxns
281
282     def setDataStyles(self, styles: list):
283         self.lineStyles = styles
284     def setDataColors(self, colors=None):

```

```

285     colors = []
286     for i in range (len(self.data)):
287         color = "#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
288         colors.append(color)
289         # print(color)
290         # print(colors)
291
292     self.dataColors = colors
293     self.setLRegLineColors(colors)
294
295     def setAxisLabels(self, x:str, y:str, indepVar=0, xpadding=5, ypadding=5):
296         self.figure.axis[indepVar].set_xlabel(x,labelpad=xpadding)
297         self.figure.axis[indepVar].set_ylabel(y,labelpad=ypadding)
298
299     def setTicProps(self, _size=4, _width=1, _direction='in'):
300         self.figure.axis[0].xaxis.set_tick_params(which='major', size=_size, width=_width,
direction=_direction, top='on')
301         self.figure.axis[0].xaxis.set_tick_params(which='minor', size=_size, width=_width,
direction=_direction, top='on')
302         self.figure.axis[0].yaxis.set_tick_params(which='major', size=_size, width=_width,
direction=_direction, right='on')
303         self.figure.axis[0].yaxis.set_tick_params(which='minor', size=_size, width=_width,
direction=_direction, right='on')
304
305     def setNumTics(self, delta_x=0.1, delta_y=0.1, x_subTics=3, y_subTics=3):
306         self.figure.axis[0].xaxis.set_major_locator(mpl.ticker.MultipleLocator(delta_x))
307
308         self.figure.axis[0].xaxis.set_minor_locator(mpl.ticker.MultipleLocator(delta_x/x_subTics))
309         self.figure.axis[0].yaxis.set_major_locator(mpl.ticker.MultipleLocator(delta_y))
310
311         self.figure.axis[0].yaxis.set_minor_locator(mpl.ticker.MultipleLocator(delta_y/y_subTics))
312
313     #Plot: Features
314     def showLegend(self, x=0.01, y=0.01, width=1, height=1, _loc='lower left',
frame=True, _fontSize=10):
315         # plt.legend(bbox_to_anchor=(x,y, width, height), loc=_loc, frameon=frame,
fontSize=_fontSize)
316         plt.legend(frameon=frame, fontsize=_fontSize)
317
318     def changeFont(self, font='Alvenir', size=10, linewidth=0.9):
319         mpl.rcParams['font.family'] = font
320         plt.rcParams['font.size'] = size
321         plt.rcParams['axes.linewidth'] = linewidth
322
323     #Presentation
324     def showPlot(self):
325         "Shows the figure"
326         plt.show()
327
328     #Save
329     def savePlot(self, filename="poop.png", _dpi=900, _transparent=False,
_bbox_inches='tight'):
330         "Saves the Figure(graph) made to a file"
331         plt.savefig(filename, dpi=_dpi, transparent=_transparent,
bbox_inches=_bbox_inches)
332
333     #Close the figure, plots, and data associated with the object
334     def close(self):

```

```
334 | "Close the figure(window) that we were plotting in"  
335 | plt.close(self.figure)
```

---