```python
#Wesley Johanson
# from aifc import _Marker
# from re import I
from re import I
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.font_manager as fm
from sklearn.linear_model import LinearRegression
from scipy import stats
import random

class ChEplot:
    def __init__(self):
        self.figure=None
        self.dataLabels=None
        self.dataColors =None
        #Counting Elements
        self.numDataVars=None
        self.numDataFns=None
        self.numDataSets=None
        self.data=None
        self.fxns2plot=None

    #Data setter/getter/modifier functions
    # def loadCSV(self, filename: str,folder=None, names=None, indepVars=1, skip=0):
    #   """Loads each column of data from the CSV file into a row of a numpy
    #   array stored in self.data

    #   'names' is a list of names for the data sets in each col of the CSV"""
    #   self.data = np.loadtxt(filename, unpack=True, \
    #                                           delimiter=',',skiprows=skip)
    #   if names is not None: self.dataLabels = names
    #   self.numDataVars = indepVars
    #   self.numDataSets = len(self.data)
    #   self.numDataFns = self.numDataSets - self.numDataVars


            #Data
    def loadLabels(self):
        pass

    def loadCSV(self, filename: str, names: list, indepVars, skip=0):
        """Loads each column of data from the CSV file into a row of a numpy
        array stored in self.data

        'names' is a list of names for the data sets in each col of the CSV"""
        # if indepVars < 1 or indepVars > len(names): return
        self.data = np.loadtxt(filename, unpack=True, delimiter=',',skiprows=skip)
        # if indepVars > self.numDataSets: self.data = none; return
        self.dataLabels = names
        self.numDataVars = indepVars
        self.numDataSets = len(self.data)
        self.numDataFns = self.numDataSets - self.numDataVars

    def loadCSV_str(self, filename: str, names: list, indepVars, skip=0):
        """Loads each column of data from the CSV file into a row of a numpy
        array stored in self.data

        'names' is a list of names for the data sets in each col of the CSV"""
        # if indepVars < 1 or indepVars > len(names): return
        self.data = np.loadtxt(filename, unpack=True, delimiter=',',skiprows=skip,
dtype=str)
        # if indepVars > self.numDataSets: self.data = none; return
        self.dataLabels = names
        self.numDataVars = indepVars
        self.numDataSets = len(self.data)
        self.numDataFns = self.numDataSets - self.numDataVars
        # for col in range(0, len(self.data)):
        #   new_x = []
        #   new_y = []
        #   for row in range(0, len(self.data[0])):
        #       if self.data[col,row] != "":
        #           new_x.append(float(self.data[col,row]))
        #           new_y.append(float(self.data[0,row]))




    def setData(self, data: list, vars=1):
        "Replaces Data and performs same operations as loadCSV"
        self.data = data
        self.numDataVars = vars
        self.numDataSets = len(self.data)
        self.numDataFns = self.numDataSets - self.numDataVars


    #Printers
    def printAllData(self):
        print(
            "\n",self.figure
            ,"\n",self.dataLabels
            ,"\n",self.numDataVars
            ,"\n",self.numDataFns
            ,"\n",self.numDataSets
            ,"\n",self.data
            ,"\n",self.fxns2plot)
        pass

    def printData(self):
        "print all data points in self.data"
        print(self.data)

    def printMeans(self):
        "Prints the  mean value of each row vector in self.data"
        for i in range(0, self.numDataSets):
            outputStr = "the mean of "
            if self.dataLabels[i] is not None:
                outputStr += self.dataLabels[i]
            outputStr += "\t\tis " + str(np.mean(self.data[i]))

    #Setters
    def setDataLabel(self, names):
        """
        Stores a list of strings into the instance, where each str in the list
        is the name of the corresponding column in the CSV file
        """
        self.dataLabels = names

    def segmentData(self):
        pass

    def setLRegLineColors(self, colors=[]):
        self.LRegLineColors = colors

    def setIndepVars(self, vars):
        "Vars are the first arrays in the self.data matrix"
        self.setIndepVars = vars

    #Plotters
    def plotData(self, width, height, markers=None):
        self.figure = plt.figure(figsize=(width, height))
        L, B, W, H = [0.15, 0.1, 0.80, 0.85]
        self.figure.axis = []
        self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
        #find a way to exclude data
        # for var in range(0, self.numDataVars):
        var = self.fxns2plot[0]
        for fn in self.fxns2plot[1:]:
            x = self.data[var]
            y = self.data[fn]
            #LOOK
            lbl = self.dataLabels[fn]
            # random_color=list(np.random.choice(range(255),size=3))
            if markers is not None:
                mk = markers[fn]
            else:
                mk = "."
            if self.dataColors is not None:
                clr = self.dataColors[fn]
                self.figure.axis[0].plot(x,y,mk,label=lbl,color=clr)
            else:
                self.figure.axis[0].plot(x,y,mk,label=lbl)

    def plotData_str(self, width, height, markers=None):
        self.figure = plt.figure(figsize=(width, height))
        L, B, W, H = [0.15, 0.1, 0.80, 0.85]
        self.figure.axis = []
        self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
        #find a way to exclude data
        # for var in range(0, self.numDataVars):
        var = self.fxns2plot[0]
        for fn in self.fxns2plot[1:]:
            x = self.data[var]
            y = self.data[fn]
            #LOOK
            lbl = self.dataLabels[fn]
            # random_color=list(np.random.choice(range(255),size=3))
            for col in range(0, len(self.data)):
                new_x = []
                new_y = []
                for row in range(0, len(self.data[0])):
                    if self.data[col,row] != "":
                        new_x.append(float(self.data[var,row]))
                        new_y.append(float(self.data[col,row]))
            if markers is not None:
                mk = markers[fn]
            else:
                mk = "."
            if self.dataColors is not None:
                clr = self.dataColors[fn]
                self.figure.axis[0].plot(new_x,new_y,mk,label=lbl,color=clr)
            else:
                self.figure.axis[0].plot(new_x,new_y,mk,label=lbl)

    def plotData_str_logx(self, width, height, markers=None):
        self.figure = plt.figure(figsize=(width, height))
        L, B, W, H = [0.15, 0.1, 0.80, 0.85]
        self.figure.axis = []
        self.figure.axis.append(self.figure.add_axes([L, B, W, H]))
        #find a way to exclude data
        # for var in range(0, self.numDataVars):
        var = self.fxns2plot[0]
        for fn in self.fxns2plot[1:]:
            x = self.data[var]
            y = self.data[fn]
            #LOOK
            lbl = self.dataLabels[fn]
            # random_color=list(np.random.choice(range(255),size=3))
            for col in range(0, len(self.data)):
                new_x = []
                new_y = []
                for row in range(0, len(self.data[0])):
                    if self.data[col,row] != "":
                        new_x.append(float(self.data[var,row]))
                        new_y.append(float(self.data[col,row]))
            if markers is not None:
                mk = markers[fn]
            else:
                mk = "."
            if self.dataColors is not None:
                clr = self.dataColors[fn]
                self.figure.axis[0].plot(new_x,new_y,mk,label=lbl,color=clr)
            else:
                self.figure.axis[0].plot(new_x,new_y,mk,label=lbl)
        self.figure.axis[0].set_xscale('log')
        # self.figure.axis[0].set_yscale('log')

    def plotLRegLines(self, width=0.5, style='-'):
        var = self.fxns2plot[0]
        fxns = self.fxns2plot[1:]
        for fn in fxns:
            m, b = np.polyfit(self.data[var],self.data[fn],1)
            x = np.linspace(min(self.data[var]), max(self.data[var]),
num=len(self.data[var]))
            y = m * x + b
            txt1 = "LinReg line for y = " + self.dataLabels[fn]
            txt2 = "and x = " + self.dataLabels[var]
            txt3 = "y = (%.4f" % m + ")x + (%.4f" % b + ")"
            txt4 = "R^2 = %.4f" % ChEplot.rSquared(self.data[var], self.data[fn])
            print( f"{txt1:<35}{txt2:<30}{txt3:>20}{txt4:>20}")
            if self.dataColors is None:
                self.figure.axis[0].plot(x, y, \
                    linewidth=width ,linestyle=style)
            else:
                self.figure.axis[0].plot(x, y, \
                    color=self.dataColors[fn], \
                    linewidth=width ,linestyle=style)

    def plotErrorBars(self):
        # stdDev =
        pass

    #Statistics
    @staticmethod
    def rSquared(x, y):
        x = x.reshape((-1,1))
        y_reg = LinearRegression().fit(x,y)
        return y_reg.score(x,y)

    def printAllRSquared(self, precision=5, vars=None, fxns=None):
        if vars is None:
            vars = range(0, self.numDataSets)
        if fxns is None:
            fxns = range(0, self.numDataSets)

        for fn in fxns:
            for var in vars:
                if fn == var:
                    continue
                rSquared = ChEplot.rSquared(self.data[0],self.data[fn])
                rStr = "R^2 = %1." + str(precision) + "f"
                rStr = rStr % rSquared
                if rStr is None: print("Error_0")
                if self.dataLabels is None: print("Error_1")
                print( f"{rStr:<25}{self.dataLabels[fn-self.numDataVars]:<20}{'with
respect to':<20}{self.dataLabels[var]:>10}")

    #COMPLETE ME
    def confInterv(self, n=1):
        self.lowerBound_CI = []
        self.upperBound_CI = []
        for fn in range(self.numDataVars, self.numDataSets):
            x_bar , stdDev = np.mean(self.data[fn]) ,np.std(self.data[fn])
            SE = stdDev / math.sqrt(self.num)
            DoF = n
            stats.t.ppf(q=0.05,)
            scipy.stats.t.ppf(q=.05,df=22)


    #Plot: Setters
    def setFxns2Plot(self, fxns):
        self.fxns2plot = fxns

    def setDataStyles(self, styles: list):
        self.lineStyles = styles
    def setDataColors(self, colors=None):
```

```python
        colors = []
        for i in range (len(self.data)):
            color = "#"+''.join([random.choice('0123456789ABCDEF') for j in range(6)])
            colors.append(color)
            # print(color)
            # print(colors)

        self.dataColors = colors
        self.setLRegLineColors(colors)

    def setAxisLabels(self, x: str, y:str, indepVar=0, xpadding=5, ypadding=5):
        self.figure.axis[indepVar].set_xlabel(x,labelpad=xpadding)
        self.figure.axis[indepVar].set_ylabel(y,labelpad=ypadding)

    def setTicProps(self, _size=4, _width=1, _direction='in'):
        self.figure.axis[0].xaxis.set_tick_params(which='major', size=_size, width=_widt
direction=_direction, top='on')
        self.figure.axis[0].xaxis.set_tick_params(which='minor', size=_size, width=_widt
direction=_direction, top='on')
        self.figure.axis[0].yaxis.set_tick_params(which='major', size=_size, width=_widt
direction=_direction, right='on')
        self.figure.axis[0].yaxis.set_tick_params(which='minor', size=_size, width=_widt
direction=_direction, right='on')

    def setNumTics(self, delta_x=0.1, delta_y=0.1, x_subTics=3, y_subTics=3):
        self.figure.axis[0].xaxis.set_major_locator(mpl.ticker.MultipleLocator(delta_x))

self.figure.axis[0].xaxis.set_minor_locator(mpl.ticker.MultipleLocator(delta_x/x_subTics
        self.figure.axis[0].yaxis.set_major_locator(mpl.ticker.MultipleLocator(delta_y))

self.figure.axis[0].yaxis.set_minor_locator(mpl.ticker.MultipleLocator(delta_y/y_subTics

    #Plot: Features
    def showLegend(self, x=0.01, y=0.01, width=1, height=1, _loc='lower left',
frame=True,_fontSize=10):
        # plt.legend(bbox_to_anchor=(x,y, width, height), loc=_loc, frameon=frame,
fontsize=_fontSize)
        plt.legend(frameon=frame, fontsize=_fontSize)


    def changeFont(self, font='Alvenir', size=10, linewidth=0.9):
        mpl.rcParams['font.family'] = font
        plt.rcParams['font.size'] = size
        plt.rcParams['axes.linewidth'] = linewidth

    #Presentation
    def showPlot(self):
        "Shows the figure"
        plt.show()

    #Save
    def savePlot(self, filename="poop.png", _dpi=900, _transparent=False,
_bbox_inches='tight'):
        "Saves the Figure(graph) made to a file"
        plt.savefig(filename, dpi=_dpi, transparent=_transparent,
bbox_inches=_bbox_inches)

    #Close the figure, plots, and data associated with the object
    def close(self):
        "Close the figure(window) that we were plotting in"
        plt.close(self.figure)
```