*#Below are the imports of plugins. Will delete ones that are not needed at the end.  
  
import* numpy *as* np  
*import* scipy *as* sp  
*import* matplotlib *as* plot  
*import* json *as* js  
*import* pandas *as* pd  
*from* matplotlib *import* pyplot *as* plt  
*import* os  
*import* tkinter *as* tk  
*from* tkinter *import* filedialog  
*import* statistics *as* st  
*import* math  
*from* math *import* sqrt   
*from* statistics *import* mean  
*import* sklearn.linear\_model  
*from* sklearn.linear\_model *import* LinearRegression  
*import* seaborn *as* sns; sns.set()  
  
*# Prevents Errors when reading a file  
#pre = os.path.dirname(os.path.realpath(\_\_file\_\_))  
  
# GUI calling CSV file*root = tk.Tk() *# Allows for GUIs to be created & manipulated*canvas1 = tk.Canvas() *# Creates canvas for GUI*canvas1.pack() *# Creates and opens GUI*import\_file\_path = filedialog.askopenfilename() *# Opens file by name via GUI*df = pd.read\_excel(import\_file\_path) *# Reads CSV file*print('\n' + 'The data frame from the CSV file:' + '\n') *# Description of info for user*print(df) *# Prints dataframe of CSV file*print('-------------------------------------------------------------------------') *# Divider for organization*print('\n' + 'The Instrumental Magnitude columns from the data frame:' + '\n') *# Description of info for user*magCols = [df *for* df *in* df.columns *if* 'Instrument' *in* df] *# Only calls cols from dataframe with 'Instrument' in name*print(list(df.columns)) *# Prints list of column names from dataframe*print('\n') *# Line of space to make things easier to view*print(magCols) *# Prints name of instrument cols*print('\n') *# Line of space to make things easier to view*dfMagCols = df.filter(regex='Instrument') *# Sets the cols w/ 'instrument' name as a variable to print*print(dfMagCols) *# Prints out new dataframe of just Instr. mag. cols.*print('-------------------------------------------------------------------------') *# Divider for organization.*print('\n' + 'The Centroid Magnitude columns from the data frame without the NaN rows:' + '\n') *# Description of info for user*dfFinalMagCols = dfMagCols.dropna() *# Drops NaNs from dataframe to get means for each col.*row, col = dfMagCols.shape *# Getting number of actual cols for dfMagCols*print(dfFinalMagCols) *# Prints out centroid mag. dataframe w/out "NaN" rows*print('-------------------------------------------------------------------------') *# Divider for organization*print('\n' + 'The means for each of the respective columns:' + '\n') *# Description of info for user*meanMagCols = dfFinalMagCols.mean() *# Gets mean of each column in final centroid mag. dataframe*print(meanMagCols) *# Prints mean of each column in final centroid mag. dataframe*print('-------------------------------------------------------------------------') *# Divider for organization.  
  
# NOTE: The y part below is for a user input for the measured/known magnitudes from Vizier.*root = tk.Tk() *# Allows for GUIs to be created & manipulated*canvas1 = tk.Canvas() *# Creates canvas for GUI*canvas1.pack() *# Creates and opens GUI*import\_file\_path = filedialog.askopenfilename() *# Opens file by name from GUI*df2 = pd.read\_excel(import\_file\_path) *# Reads CSV file imported*yVars = [] *# Creates empty array of y-variables*yVars.append(df2) *# Creates array of y-variables w/ vals. from df2 (newly imported csv file)*x = meanMagCols *# Renames meanMagCols as x*y = yVars *# Renames yVars as y*x = np.array(x) *# Makes x into a numpy array*y = np.array(y) *# Makes y into a numpy array*y = y.flatten() *# Flattens y so program can do calculations w/ it*print('Here are the magnitudes for each star! \n') *# Tells user what's being shown*print("The Instrumental Magnitudes: \n", x, "\n") *# Presents instrumental mags from raw data*print("The magnitudes from VizieR: \n", y) *# Presents mags user looked up for reference on VizieR  
  
  
# First runthrough of Linear Regression of Data:*x = np.array(x).reshape((-1, 1)) *# Reshapes array to be used for calculations... making this new x*linReg = LinearRegression() *# Renaming LinearRegression() as linReg*linReg = linReg.fit(x, y) *# Making Python perform linear regression on values for x & y*r\_sq = linReg.score(x, y) *# Obtaining the coefficient of determination (AKA R^2)*print('The coefficient of determination (R^2): \n', r\_sq, '\n') *# Printing R^2*print('Y-intercept: \n', linReg.intercept\_, '\n') *# Printing the y-int. for the linear regression*newLinReg = sklearn.linear\_model.LinearRegression().fit(x, y.reshape( (-1, 1))) *# Reshaping y data so slope can be calculated*slope = newLinReg.coef\_ *# Obtaining slope of regression line*slopeFlat = slope.flatten() *# Makes it into a 1D array... less confusing than 2D array*print('Slope:\n', slopeFlat, '\n') *# Printing 1D array of the slope*y\_pred = linReg.predict(x) *# Using linear regression to predict y-values*y\_pred = linReg.intercept\_ + linReg.coef\_ \* x *# Using linear regression to predict y-values*y\_predFlat = y\_pred.flatten() *# Flatten array for predicted y-values*print('Predicted response:', y\_predFlat, '\n', sep='\n') *# Printing 1D array for predicted y-values*resid = x - y\_pred *# Calculating residuals by subtracting x-values by predicted y-values*residFlat = resid.flatten() *# Flattening array for residuals*print('This is the residual for each point: \n', residFlat, '\n') *# Printing flattened residual array*range1 = range(1, col+1, 1) *# Setting range for number of columns*annotations = list(range1) *# Getting list of range we set up and calling it "annotations"*plt.figure(figsize=(8, 6)) *# ??? (LOOK UP WHAT THIS DOES!!!!)*plt.title("Scatter Plot with annotations", fontsize=15) *# Adds a title to the plot  
  
for* i, label *in* enumerate(annotations): *# For loop to label each point plotted in order from 1 to N* plt.annotate(label, (x[i], y[i]))  
  
x\_new = np.arange(20).reshape((-1, 1)) *# Obtaining new x-values via linear regression*y\_new = linReg.predict(x\_new) *# Obtaining new y-values via linear regression*plt.plot(x\_new, y\_new, color='#68B4E7'); *# Line of best fit*plt.scatter(x, y, color='#F7514E') *# Scatter plot of original data*plt.scatter(x, y\_pred, color='#15C551') *# Predicted data*plt.title('Instr. Mag. vs Exp. Mag.') *# Title*plt.xlabel('Instrumental Magnitude') *# x-axis label*plt.ylabel('Expected Magnitude') *# y-axis label*plt.show() *# Telling Python to plot the data  
  
# Run-through of Linear Regression of data w/ omitted star...*starRemInput = input("Enter the list of potential variable stars with a space in between each one:" + "\n")  
  
starRem = starRemInput.split() *# Taking the user input and turning it into a list  
  
# convert each item to int type  
for* i *in* range(len(starRem)):  
 *# convert each item to int type* starRem[i] = int(starRem[i])  
  
starRem = np.array(starRem) *# Taking the list from the user input and turing it into an array*starRemIndex = starRem - 1 *# Translating the numbers into the array of indices to omit the datapoint in their indices*print("\n") *# Printing a space for aesthetic reasons*x2 = np.delete(x, starRemIndex) *# Deleting star based off of user input array*y2 = np.delete(y, starRemIndex) *# Deleting star based off of user input array*x2 = np.array(x2).reshape((-1, 1)) *# Reshapes the array so it can be used for calculations... naming this x2*y2 = np.array(y2) *# Making y2 the variable to represent numpy array of y-values*linReg2 = linReg.fit(x2, y2) *# Making Python perform linear regression on values for x2 and y2*r\_sq2 = linReg.score(x2, y2) *# Obtaining coefficient of determination (AKA R^2)*print('The coefficient of determination (R^2): \n', r\_sq2, '\n') *# Printing R^2*print('Y-intercept: \n', linReg2.intercept\_, '\n') *# Printing the y-int. for the linear regression*newLinReg2 = sklearn.linear\_model.LinearRegression().fit(x2, y2.reshape((-1, 1))) *# Reshaping y2 data so slope can be calculated*slope2 = newLinReg2 *# Obtaining slope of regression line*print('slope:', slope2.coef\_) *# Printing 1D array of slope*y2\_pred = linReg2.predict(x2) *# Using linear regression to predict y-values*y2\_pred = linReg2.intercept\_ + linReg2.coef\_ \* x2 *# Using linear regression to predict y-values*y\_predFlat2 = y2\_pred.flatten() *# Flattening array for predicted y-values*print('Predicted response:', y\_predFlat2, '\n', sep='\n') *# Printing 1D array for predicted y-values*resid2 = x2 - y2\_pred *# Calculating residuals by subtracting x-values by predicted y-values*residFlat2 = resid2.flatten() *# Flattening array for residuals*print('This is the residual for each point: \n', residFlat2, '\n') *# Printing flattened residual array*x2\_new = np.arange(20).reshape((-1, 1)) *# Obtaining new x-values via linear regression*y2\_new = linReg2.predict(x2\_new) *# Obtaining new y-values via linear regression*slope2 = newLinReg2.coef\_ *# Slope for line*slopeFlat2 = slope2.flatten() *# Flattening of slope*print('Slope:\n', slopeFlat2, '\n') *# Printing flattened slope*YInt2 = linReg2.intercept\_ *# Getting y-int. of slope*print("This is the new y-int.: \n ", YInt2, "\n") *# Printing y-int. of slope*plt.plot(x2\_new, y2\_new, color='#68B4E7', label='Line of Best Fit') *# Plotting line of best fit*plt.scatter(x2, y2, color='#F7514E', label='Real Values') *# Plotting actual values*plt.scatter(x2, y2\_pred, color='#15C551', label='Predicted y-values') *# Plotting predicted values*plt.title('Instr. Mag. vs Exp. Mag. (W/out Var. Star)') *# Title of the plot*plt.xlabel('Instrumental Magnitude') *# Label for x-value*plt.ylabel('Expected Magnitude') *# Label for y-value*plt.show() *# Having Python show plot  
  
  
# Runthrough of Linear Regression of Data w/ Error Bars*x = np.array(x).reshape((-1, 1)) *# Reshapes array to be used for calculations... making this new x*y = np.array(y) *# Creates an array of the y-values*linReg = LinearRegression() *# Renaming LinearRegression() as linReg*linReg.fit(x, y) *# Making Python perform linear regression on values for x & y*linReg = LinearRegression().fit(x, y) *# Making Python perform linear regression on values for x & y*r\_sq = linReg.score(x, y) *# Obtaining the coefficient of determination (AKA R^2)*print('The coefficient of determination (R^2): \n', r\_sq, '\n') *# Printing R^2*print('Y-intercept: \n', linReg.intercept\_, '\n') *# Printing the y-int. for the linear regression*newLinReg = sklearn.linear\_model.LinearRegression().fit(x, y.reshape((-1, 1))) *# Reshaping y data so slope can be calculated*slope = newLinReg.coef\_ *# Obtaining slope of regression line*slopeFlat = slope.flatten() *# Makes it into a 1D array... less confusing than 2D array*print('Slope:\n', slopeFlat, '\n') *# Printing 1D array of the slope*y\_pred = linReg.predict(x) *# Using linear regression to predict y-values*y\_pred = linReg.intercept\_ + linReg.coef\_ \* x *# Using linear regression to predict y-values*y\_predFlat = y\_pred.flatten() *# Flatten array for predicted y-values*print('Predicted response:', y\_predFlat, '\n', sep='\n') *# Printing 1D array for predicted y-values*resid = x - y\_pred *# Calculating residuals by subtracting x-values by predicted y-values*residFlat = resid.flatten() *# Flattening array for residuals*print('This is the residual for each point: \n', residFlat, '\n') *# Printing flattened residual array*x\_new = np.arange(20).reshape((-1, 1)) *# Obtaining new x-values via linear regression*y\_new = linReg.predict(x\_new) *# Obtaining new y-values via linear regression*xFlat = x.flatten() *# Flattening the x-values*calcMag = slope2 \* xFlat + YInt2 *# Obtaining the calculated magnitudes*yerr = y - calcMag *# Subtracting y by the calculated magnitude to get the error in the y-dir.*yerr = yerr.flatten() *# Flattening the error in the y-dir.*yerr = yerr \*\* 2 *# Squaring the error in the y-dir.*yerr = yerr.mean() *# Obtaining the mean if the error in the y-direction*stndrdErrOfEst = sqrt(yerr / col) *# Obtaining the standard error of estimate to use to plot the error bars*print("Standard Error of the Estimate: \n", stndrdErrOfEst) *# Printing the Standard Error of Estimate*range1 = range(1, col+1, 1) *# Setting range for number of columns*annotations = list(range1) *# Getting list of range we set up and calling it "annotations"*plt.figure(figsize=(8, 6)) *#*plt.title("Scatter Plot with annotations", fontsize=15) *# Adds a title to the plot  
  
for* i, label *in* enumerate(annotations): *# For loop to label each point plotted in order from 1 to N* plt.annotate(label, (x[i], y[i]))  
  
plt.plot(x\_new, y\_new); *# Plotting the new x-values against the new y-values*plt.scatter(x, y) *# Making a scatter plot*plt.scatter(x, y, label='Real Values') *# Plotting the original (real) x and y values*plt.plot(x\_new, y\_new, color='#68B4E7', label='Line of Best Fit'); *# Plotting the new x and y values*plt.scatter(x, y\_pred, color='#15C551', label='Predicted y-values') *# Making the scatter plot of the predicted y-values*plt.gca().legend() *# Creates a legend next to the graph*plt.errorbar(x, y, color='#F7514E', yerr=stndrdErrOfEst, fmt="o", ecolor='black', elinewidth=2, capsize=2) *# Makes scatter plot of error bars*plt.title('Instr. Mag. vs Exp. Mag. w/ Error Bars') *# Title of the graph*plt.xlabel('Instrumental Magnitude') *# Label for the x-axis of the plot*plt.ylabel('Expected Magnitude') *# Label for the y-axis of the plot*plt.show() *# Shows the plot  
  
# End of code*