```python
import numpy as np
from matplotlib import pyplot as plt

def scatter_plot(x, y, x_error, y_error, x_label, y_label, title, color, trendline):
    '''
    plots a scatter plot including error bars

    Parameters
    ----------
    x : list
        list of x values for the graph to plot. must be the same length as y.
    y : list
        list of y values for the graph to plot. must be the same length as x.
    x_error : list/bool
        list of errors for the x values. must be the same length as x and y. can be boolean False if there are no x errors.
    y_error : list/bool
        list of errors for the y values. must be the same length as x and y. can be boolean False if there are no y errors.
    x_label : str
        label to display on the x axis of the graph.
    y_label : str
        label to display on the y axis of the graph.
    title : str
        label to display as the title of the graph.
    color : str
        colour of the plotted points of the graph.
    trendline : bool
        True to plot trendline. False to not plot trendline.

    Returns
    -------
    line_fit : list
        a list of two values, the first position is gradient and the second position is y intercept.
    '''
    plt.ylabel(y_label) # set the y axis label
    plt.xlabel(x_label) # set the x axis label
    plt.title(title) # set the title

    plt.errorbar(x, y, yerr = y_error, xerr = x_error, fmt = 's', ecolor = 'black', markersize = 3, capsize = 3, linewidth = 0.7, c = color) # configure the er

    if trendline == True: # if the trendline is set to be drawn
        line_fit = np.polyfit(x, y, 1) # find the gradient and intercept
        p = np.poly1d(line_fit) # set up a function using the gradient and intercept
        plt.plot(x, p(x), c = color, linewidth = 1) # plot a line with the gradient and intercept
        return line_fit # returns the line of best fit

def scatter_plot_with_gradient_uncertainty(x, y, x_error, y_error, x_label, y_label, title, model, gradient_range, intercept_range):
    '''
    plots a scatter plot including error bars and a light gray area of gradient uncertainty

    Parameters
    ----------
    x : list
        list of x values for the graph to plot. must be the same length as y.
    y : list
        list of y values for the graph to plot. must be the same length as x.
    x_error : list/bool
        list of errors for the x values. must be the same length as x and y. can be boolean False if there are no x errors.
    y_error : list/bool
        list of errors for the y values. must be the same length as x and y. can be boolean False if there are no y errors.
    x_label : str
        label to display on the x axis of the graph.
    y_label : str
        label to display on the y axis of the graph.
    title : str
        label to display as the title of the graph.
    model : list
        a list of two values, the first position is gradient and the second position is y intercept.
    gradient_range : list
        a list of two values, the first position is lowest possible gradient and the second position is highest possible gradient.
    intercept_range : list
        a list of two values, the first position is lowest possible intercept and the second position is highest possible intercept.

    Returns
    -------
    None.
    '''
    plt.ylabel(y_label) # set the y axis label
    plt.xlabel(x_label) # set the x axis label
    plt.title(title) # set the title

    gradient_list = np.linspace(gradient_range[0], gradient_range[1], num=50) # make a list of gradients between the minimum and maximum gradient
    intercept_list = np.linspace(intercept_range[0], intercept_range[1], num=50) # make a list of intercepts between the minumum and maximum intercepts

    for i in gradient_list: # for all the possible gradients
        for j in intercept_list: # and all the possible intercepts
            px = x*i+j # makes function
            plt.plot(x, px, c = 'lightgrey', linewidth = 1, alpha = 1) # plots all the gradients in light gray

    plt.errorbar(x, y, yerr = y_error, xerr = x_error, fmt = 's', ecolor = 'black', markersize = 5, capsize = 3, linewidth = 0.7) # configures the error bars
    px = model[0]*(x)+model[1] # makes function
    plt.plot(x, px, c = 'blue', linewidth = 1) # plots the line of best fit

    plt.show() # reveals plot

def chi_square(x, y, x_error, y_error, model):
    '''
    function which takes a model and experimental values and gives the chi squared value of that model (in this case reduced chi squared due to the inputs)

    Parameters
    ----------
    x : list
        list of x values for the graph to plot. must be the same length as y.
    y : list
        list of y values for the graph to plot. must be the same length as x.
    x_error : list/bool
        list of errors for the x values. must be the same length as x and y. can be boolean False if there are no x errors.
```

```
    y_error : list/bool
        list of errors for the y values. must be the same length as x and y. can be boolean False if there are no y errors.
    model : list
        a list of two values defining the model, the first position is gradient and the second position is y intercept.

    Returns
    -------
    chi_square_value : float
        value of (normal/reduced depending on the input) chi squared for the model.
    '''
    observed = y # sets y as the observed value
    expected = x * model[0] + model[1] # finds the value predicted at that position by the model

    chi_square_value = sum( ( (observed - expected) ** 2) / ( ( model[0] ** 2 ) * (x_error ** 2) + ( y_error ** 2) ) ) # finds the chi-squared value of fitness

    return chi_square_value

def scalar_matrix(x, y, x_error, y_error, gradient_min, gradient_max, intercept_min, intercept_max):
    '''
    creates the 2d matrix of reduced chi squared values by testing many models to the data.

    Parameters
    ----------
    x : list
        list of x values for the graph to plot. must be the same length as y.
    y : list
        list of y values for the graph to plot. must be the same length as x.
    x_error : list/bool
        list of errors for the x values. must be the same length as x and y. can be boolean False if there are no x errors.
    y_error : list/bool
        list of errors for the y values. must be the same length as x and y. can be boolean False if there are no y errors.
    gradient_min : int
        lowest gradient to test in the model.
    gradient_max : int
        highest gradient to test in the model.
    intercept_min : int
        lowest intercept to test in the model.
    intercept_max : int
        highest intercept to test in the model.

    Returns
    -------
    matrix : array
        a matrix of reduced chi squared value matrix with gradient on the x axis and intercept on the y axis.
    gradient_list : list
        list of all gradients used across the x axis of the matrix.
    intercept_list : list
        list of all intercepts used across the y axis of the matrix.
    '''
    gradient_list = np.linspace(gradient_min,gradient_max,num=300) # makes a list of numbers from the minimum gradient to the maximum gradient
    intercept_list = np.linspace(intercept_min,intercept_max,num=300) # makes a list of numbers from the minimum intercept to the maximum intercept
    matrix = np.zeros((300,300)) # makes a matrix with dimensions the same size as the lists of gradients and intercepts

    for i in range(len(gradient_list)): # for every gradient in the list
        for j in range(len(intercept_list)): # and for every intercept in the list
            fit = [gradient_list[i], intercept_list[j]] # makes a new model based on the gradient and intercept
            matrix[j][i] = chi_square(x, y, x_error, y_error, fit) # checks how well the model fits the data and then sets the position in the matrix to the ch

    return matrix, gradient_list, intercept_list

def contour_plot(matrix, gradient_list, intercept_list, num):
    '''
    plots a filled contour plot in this case of the chi squared values for each gradient and intercept model tested

    Parameters
    ----------
    matrix : array
        a matrix of reduced chi squared value matrix with gradient on the x axis and intercept on the y axis.
    gradient_list : list
        list of all gradients used across the x axis of the matrix.
    intercept_list : list
        list of all intercepts used across the y axis of the matrix.
    num : int
        number of contours used in the plot.

    Returns
    -------
    None.
    '''
    plt.close() # closes previous plot
    plt.title('Contour Plot of Reduced X\N{SUPERSCRIPT TWO} for Possible Models') # sets title
    plt.ylabel('Intercept') # sets y axis label
    plt.xlabel('Gradient') # sets x axis label
    plt.contourf(gradient_list, intercept_list, np.log10(matrix), num, cmap = 'RdGy') # makes a filled contour graph using the matrix's chi-squared values
    cbar = plt.colorbar() # adds a key bar to help understand the colours on the graph
    cbar.set_label('log\N{SUBSCRIPT ONE}\N{SUBSCRIPT ZERO}(X\N{SUPERSCRIPT TWO})') # sets the label on the colour bar

def matrix_search(matrix, chi_squared_deviation, gradient_min, gradient_max, intercept_min, intercept_max):
    '''
    searches through the matrix of reduced chi squared values to find the values within the standard deviation to find
    the error of the parameters and then visualises it with a box on the graph

    Parameters
    ----------
    matrix : array
        a matrix of reduced chi squared value matrix with gradient on the x axis and intercept on the y axis.
    chi_squared_deviation : float
        value of standard deviation of reduced chi squared given by sqrt(2/v) where v is the degrees of freedom (no. of datapoints - model parameters).
    gradient_min : int
        lowest gradient to test in the model.
    gradient_max : int
        highest gradient to test in the model.
    intercept_min : int
        lowest intercept to test in the model.
    intercept_max : int
        highest intercept to test in the model.
```

```
    Returns
    -------
    gradient_error_list : list
        list of two values with errors in the gradient, negative error then positive error.
    intercept_error_list : list
        list of two values with errors in the intercept, negative error then positive error.
    best_fit : list
        a list of two values defining a model, the first position is gradient and the second position is y intercept.
    gradient_range : list
        list of the smallest value the gradient could take, then the largest value the gradient could take.
    intercept_range : list
        list of the smallest value the intercept could take, then the largest value the intercept could take.
    '''
    gradient_list = np.linspace(gradient_min,gradient_max,num=300) # makes a list of numbers from the minimum gradient to the maximum gradient
    intercept_list = np.linspace(intercept_min,intercept_max,num=300) # makes a list of numbers from the minimum intercept to the maximum intercept

    acceptable_gradient = [] # sets up the list of acceptable gradients
    acceptable_intercept = [] # sets up the list of acceptable intercepts

    lowest_chi_squared = 1000 # sets the initial lowest chi squared as a high number to be decreased
    for i in range(len(gradient_list)): # for all possible gradients
        for j in range(len(intercept_list)): # and for all possible intercepts
            if matrix[j][i] < lowest_chi_squared: # if the new value being checked is lower than the current lowest chi squared value
                lowest_chi_squared = matrix[j][i] # make the lowest chi squared value the new value
                best_fit = [gradient_list[i], intercept_list[j]] # set the best fit the the position in the matrix where the lowest chi squared value is

    max_chi_squared = chi_squared_deviation # set the chi squared deviation as the maximum chi squared value

    for i in range(len(gradient_list)): # for all gradients
        for j in range(len(intercept_list)): # and for all intercepts
            if matrix[j][i] <= max_chi_squared: # if the matrix value is within the standard deviation
                acceptable_gradient = np.append(acceptable_gradient, gradient_list[i]) # set that gradient as an acceptable gradient
                acceptable_intercept = np.append(acceptable_intercept, intercept_list[j]) # set that intercept as an acceptable intercept

    np.sort(acceptable_gradient) # sort the acceptable gradient list in ascending order
    gradient_range = [acceptable_gradient[0], acceptable_gradient[len(acceptable_gradient) - 1]] # choose the first and last gradients in the list

    np.sort(acceptable_intercept) # sort the acceptable intercept list in ascending order
    intercept_range = [acceptable_intercept[0], acceptable_intercept[len(acceptable_intercept) - 1]] # choose the first and last intercepts in the list

    gradient_error_list = np.sort( - (best_fit[0] - gradient_range)) # find the distance of the gradient ranges from the best fit
    intercept_error_list = np.sort( - (best_fit[1] - intercept_range)) # find the distance of the intercept ranges from the best fit

    contour_plot(matrix, gradient_list, intercept_list, 35) # draw a contour plot using the contour plot function
    plt.plot([gradient_range[0], gradient_range[0]], [intercept_range[0], intercept_range[1]], color = 'black') # draw part of a box around the acceptable part
    plt.plot([gradient_range[1], gradient_range[1]], [intercept_range[0], intercept_range[1]], color = 'black') # draw part of a box around the acceptable part
    plt.plot([gradient_range[0], gradient_range[1]], [intercept_range[0], intercept_range[0]], color = 'black') # draw part of a box around the acceptable part
    plt.plot([gradient_range[0], gradient_range[1]], [intercept_range[1], intercept_range[1]], color = 'black') # draw part of a box around the acceptable part
    plt.show() # reveal the plot

    return gradient_error_list, intercept_error_list, best_fit, gradient_range, intercept_range

def chi_square_test(x, y, x_error, y_error, gradient_min, gradient_max, intercept_min, intercept_max, x_label, y_label, title):
    '''
    handles the chi square test to find the parameters for best fit and the errors in those parameters

    Parameters
    ----------
    x : list
        list of x values for the graph to plot. must be the same length as y.
    y : list
        list of y values for the graph to plot. must be the same length as x.
    x_error : list/bool
        list of errors for the x values. must be the same length as x and y. can be boolean False if there are no x errors.
    y_error : list/bool
        list of errors for the y values. must be the same length as x and y. can be boolean False if there are no y errors.
    gradient_min : int
        lowest gradient to test in the model.
    gradient_max : int
        highest gradient to test in the model.
    intercept_min : int
        lowest intercept to test in the model.
    intercept_max : int
        highest intercept to test in the model.
    x_label : str
        label to display on the x axis of the graph.
    y_label : str
        label to display on the y axis of the graph.
    title : str
        label to display as the title of the graph.

    Returns
    -------
    best_fit : list
        a list of two values defining a model, the first position is gradient and the second position is y intercept.
    gradient_error : float
        a single number representing the error in the gradient, e.g. gradient = [gradient] +/- gradient_error.
    intercept_error : float
        a single number representing the error in the intercept, e.g. intercept = [intercept] +/- intercept_error.
    '''
    degrees_of_freedom = len(x) - 2 # defines the degrees of freedom variable as number of data points minus number of model parameters
    chi_squared_standard_deviation = np.sqrt(2 / degrees_of_freedom) # defines chi squared standard deviation variable as reduced chi squared standard deviatio

    matrix, gradient_list, intercept_list = scalar_matrix(x, y, x_error, y_error, gradient_min, gradient_max, intercept_min, intercept_max) # create chi square
    matrix = matrix / degrees_of_freedom # converts chi squared matrix into reduced chi squared matrix
    contour_plot(matrix, gradient_list, intercept_list, 35) # draws a contour plot using the contour plot function
    plt.show() # reveal the plot

    input('\nPress Enter to Continue')

    gradient_error_list, intercept_error_list, best_fit, gradient_range, intercept_range = matrix_search(matrix, chi_squared_standard_deviation, gradient_min,

    gradient_error = sum(abs(gradient_error_list)) / 2 # find the gradient error as a single number
    intercept_error = sum(abs(intercept_error_list)) / 2 # find the intercept error as a single number
```

```python
    print('\nGradient:', best_fit[0], '+/-', gradient_error) # print the gradient and the error
    print('Intercept:', best_fit[1], '+/-', intercept_error) # print the intercept and the error
    print('X\N{SUPERSCRIPT TWO} value of best fit:', chi_square(x, y, x_error, y_error, best_fit)) # print the chi squared value of best fit
    input('\nPress Enter to Continue')

    scatter_plot_with_gradient_uncertainty(x, y, x_error, y_error, x_label, y_label, title, best_fit, gradient_range, intercept_range) # plot the scatter plot

    return best_fit, gradient_error, intercept_error

def analyse_MW_data():
    '''
    analyses data from cepheid variable stars inside the milky way to model the relation between period and absolute magnitude

    Returns
    -------
    best_fit : list
        a list of two values defining a model, the first position is gradient and the second position is y intercept.
    gradient_error : float
        a single number representing the error in the gradient, e.g. gradient = [gradient] +/- gradient_error.
    intercept_error : float
        a single number representing the error in the intercept, e.g. intercept = [intercept] +/- intercept_error.
    '''
    parallax, parallax_error, period, magnitude, extinction, extinction_error = np.loadtxt('MW_Cepheids.dat',unpack = True, dtype = float, usecols = range(1,7)

    distance = 1000/parallax # find distance in parsecs from parallax
    distance_error = np.abs( ( - 2000 / ( parallax ** 2 ) ) * parallax_error ) # find error in distance through error propagation

    # M = m - 5log_10(d_pc)+5-A   absolute magnitude equation | A is extinction caused by gas and dust

    absolute_magnitude = magnitude - 5 * np.log10(distance) + 5 - extinction # absolute magnitude calculation
    absolute_magnitude_error = np.sqrt( extinction_error ** 2 + ( distance_error * 5 / ( np.log(10) * distance ) ) ** 2 ) # absolute magnitude calculation erro

    log10_period = np.log10(period) # convert period to log_10 period
    scatter_plot(log10_period, absolute_magnitude, False, absolute_magnitude_error, 'log\N{SUBSCRIPT ONE}\N{SUBSCRIPT ZERO}(Period)', 'Absolute Magnitude', 'Ce
    period_average = np.average(log10_period) # defines the period_average variable
    log10_period = log10_period - period_average # subtract average to remove correlation between parameters
    plt.show()

    print('\nEquation: M = A * log\N{SUBSCRIPT ONE}\N{SUBSCRIPT ZERO}(P) + B') # Cepheid Period-Luminosity Relationship
    input('\nPress Enter to Continue')

    best_fit, gradient_error, intercept_error = chi_square_test(log10_period, absolute_magnitude, False, absolute_magnitude_error, -4, 0, -3.8, -3.6, 'log\N{SU

    best_fit[1] = best_fit[1] + period_average # shift intercept to correct position to account for earlier when the parameter correlation was removed

    return best_fit, gradient_error, intercept_error

def analyse_ncg4527_data(PL_model_parameters, PL_A_error, PL_B_error):
    '''
    analyses the cepheid variable stars inside the ncg4527 galaxy using the previously found period-luminosity relationship

    Parameters
    ----------
    PL_model_parameters : list
        alpha parameter (gradient) and then beta parameter (intercept) of the cepheid period-luminosity relationship.
    PL_A_error : float
        error in parameter alpha (gradient) for the cepheid period-luminosity relationship.
    PL_B_error : float
        error in parameter beta (intercept) for the cepheid period-luminosity relationship.

    Returns
    -------
    ncg_distance : float
        estimated distance to ncg4527 in parsecs.
    ncg_distance_error : float
        error in the estimated distance to ncg4527 in parsecs.
    '''
    ncg_log10_period, ncg_apparent_magnitude = np.loadtxt('ngc4527_cepheids.dat',unpack = True, dtype = float, usecols = (1,2)) # load cepheid data from the ga
    ncg_extinction = 0.0682 # constant accounting for the gas and dust obstructing the galaxies light from earth

    ncg_absolute_magnitude = PL_model_parameters[0] * ncg_log10_period + PL_model_parameters[1] # calculates the absolute magnitude using the model parameters

    # M = A * log(P) + B

    ncg_absolute_magnitude_error = [] # sets up the list of absolute magnitude error

    for i in ncg_log10_period: # for all data points
        magnitude_error = np.sqrt( ( ( i * PL_A_error ) ** 2 ) + ( ( PL_B_error ) ** 2 ) ) # calculate magnitude error for each point using error propagation
        ncg_absolute_magnitude_error.append(magnitude_error) # append the error to a list of errors for the data points

    # now we have ncg_absolute_magnitude, a list of magnitudes of different stars in ncg4527
    # and we have ncg_absolute_magnitude_error, a list of errors for those magnitudes

    print('\nEquation: M = m - 5log\N{SUBSCRIPT ONE}\N{SUBSCRIPT ZERO}(d) + 5 - A') # absolute magnitude and apparent magnitude relation

    ncg_distance_list = 10 ** ( ( - ncg_absolute_magnitude + 5 - ncg_extinction + ncg_apparent_magnitude ) / (5) ) # rearrangement of absolute magnitude equati

    ncg_distance_error_list = abs( ( -(1/5) * np.log(10) * ( 10 ** ( ( 5 - ncg_absolute_magnitude - ncg_extinction + ncg_apparent_magnitude ) / 5 ) ) ) * ncg_a

    xaxis = list(range(len(ncg_distance_list))) # creates a list of length of distance list

    scatter_plot(xaxis, ncg_distance_list, False, ncg_distance_error_list, 'Cepheid No.', 'Distance (pc x 10\N{SUPERSCRIPT SEVEN})', 'Distance to Cepheids in n
    print('\nAs you can see, there is a major outlier, which I have ignored for the calculations and line of best fit')

    ncg_distance_list = np.delete(ncg_distance_list, 6) # remove outlier from distance data
    ncg_distance_error_list = np.delete(ncg_distance_error_list, 6) # remove outlier from distance error data

    ncg_distance = np.mean(ncg_distance_list) # find mean distance
    x_line = np.linspace(0,len(ncg_distance_list)+1,1001) # make list of
    mean_list = [ncg_distance] * 1001 # creates a list 1001 length where all the values are the mean distance

    plt.plot(x_line, mean_list, c = 'grey', linewidth = 1) # plots the mean line on the graph
    plt.show() # reveals the graph

    print('\nEstimated Distance to ncg4527:', ncg_distance, 'pc') # prints the estimated distance to ncg4527
```

```python
    ncg_distance_error = np.sqrt( sum( ( ncg_distance_error_list / len( ncg_distance_error_list ) ) ** 2 ) ) # error propagation for distance to ncg4527

    print('Error in Distance to ncg4527:', ncg_distance_error, 'pc') # prints the error in distance to ncg4527

    return ncg_distance, ncg_distance_error

def hubble_constant(estimated_distance, estimated_distance_error, recession_velocity):
    '''
    uses the distance and velocity data from ncg4527 to find a value for the hubble constant for the galaxy

    Parameters
    ----------
    estimated_distance : float
        estimated distance to galaxy in parsecs.
    estimated_distance_error : float
        error in the estimated distance to galaxy in parsecs.
    recession_velocity : float
        the recession velocity of the galaxy

    Returns
    -------
    hubble_constant : float
        estimation of the hubble constant for this single galaxy (km/s/Mpc).
    hubble_constant_error : float
        error in estimation of the hubble constant for this single galaxy (km/s/Mpc).
    '''
    print('\nEquation: v = H\N{SUBSCRIPT ZERO}D') # hubble constant equation

    estimated_distance_Mpc = estimated_distance / ( 10 ** 6 ) # converts distance pc to Mpc
    estimated_distance_error_Mpc = estimated_distance_error / ( 10 ** 6 ) # converts distance error pc to Mpc

    hubble_constant = recession_velocity / estimated_distance_Mpc # uses hubble constant equation to calculate hubble constant of ncg4527

    hubble_constant_error = np.sqrt( ( ( - recession_velocity / ( estimated_distance_Mpc ** 2 ) ) * estimated_distance_error_Mpc ) ** 2 ) # error propagation t

    print('\nResults for just ncg4527: ')
    print('\nHubble Constant Estimate:', hubble_constant) # estimate of the hubble constant from ncg4527
    print('Error in Hubble Constant Estimate:', hubble_constant_error) # error of the estimate of the hubble constant from ncg4527

    return hubble_constant, hubble_constant_error

def hubble_constant_multiple_galaxies(ncg_distance, ncg_distance_error, use_own_data):
    '''
    uses distance and recession velocity data from many galaxies to get a result for the hubble constant

    Parameters
    ----------
    ncg_distance : float
        estimated distance to ncg4527 in parsecs.
    ncg_distance_error : float
        error in the estimated distance to ncg4527 in parsecs.
    use_own_data : bool
        if True then the ncg4527 data will be used alongside the other galaxy data, if False then it will not.

    Returns
    -------
    result : list
        a list of two values, the estimated value of the hubble constant from the data, and then the error in the estimated hubble constant.
    '''
    recession_velocity_list, distance_list, distance_error_list = np.loadtxt('other_galaxies.dat', unpack = True, dtype = float, usecols = (1,2,3)) # load dist

    if use_own_data == True: # if it is set to use ncg4527 data
        recession_velocity_list = np.append(recession_velocity_list, 1152.0) # add 1152 to the recession velocity list
        distance_list = np.append(distance_list, ncg_distance / ( 10 ** 6 ) ) # add the ncg4527 distance to the distance list
        distance_error_list = np.append(distance_error_list, ncg_distance_error / ( 10 ** 6 ) ) # add the ncg4527 distance error to the distance error list
        print('\nUsing Data from ncg4527:')
    else: # if not using ncg4527 data
        print('\nExcluding Data from ncg4527:')

    scatter_plot(distance_list, recession_velocity_list, distance_error_list, False, 'Distance (Mpc)', 'Recession Velocity (km/s)', 'Hubble Constant Plot', 'bl
    plt.show() # reveal the plot

    input('\nPress Enter to Continue')

    if use_own_data == True: # if it is set to use ncg4527 data
        intrinsic_scatter = 4 # amount of intrinsic scatter added to the data to fit the model
    else: # if not using ncg4527 data
        intrinsic_scatter = 3 # amount of intrinsic scatter added to the data to fit the model

    distance_error_list = distance_error_list + intrinsic_scatter # add intrinsic scatter to better fit model

    scatter_plot(distance_list, recession_velocity_list, distance_error_list, False, 'Distance (Mpc)', 'Recession Velocity (km/s)', 'Hubble Constant Plot', 'bl
    plt.show() # reveals the plot

    print('\nAdded Intrinsic Scatter of +/-', intrinsic_scatter, 'Mpc') # displays how much intrinsic scatter is used
    input('\nPress Enter to Continue')

    average_distance = np.average(distance_list) # defines average distance variable
    distance_list = distance_list - average_distance # subtracts average to remove correlation between parameters

    if use_own_data == True: # if it is set to use ncg4527 data
        best_fit, gradient_error, intercept_error = chi_square_test(distance_list, recession_velocity_list, distance_error_list, False, 50, 100, 750, 1000,'Dis
    else: # if not using ncg4527 data
        best_fit, gradient_error, intercept_error = chi_square_test(distance_list, recession_velocity_list, distance_error_list, False, 50, 150, 750, 1000,'Dis

    result = best_fit[0], gradient_error # defines result using the gradient and gradient error

    return result

def Main():
    '''
    starts the process and calls all the functions needed to run the programme

    Returns
    -------
    None.
```

```
    '''
    PL_model_parameters, PL_A_errors, PL_B_errors = analyse_MW_data() # analyses data from cepheid variable stars inside the milky way to model the relation be
    input('\nPress Enter to Continue')
    ncg_distance, ncg_distance_error = analyse_ncg4527_data(PL_model_parameters, PL_A_errors, PL_B_errors) # analyses the cepheid variable stars inside the ncg
    input('\nPress Enter to Continue')
    ncg_hubble_constant, ncg_hubble_constant_error = hubble_constant(ncg_distance, ncg_distance_error, 1152.0) # uses the data from ncg4527 to find a value for
    input('\nPress Enter to Continue')
    result_including_ncg4527 = hubble_constant_multiple_galaxies(ncg_distance, ncg_distance_error, True) # uses the ncg4527 data and data from other galaxies t
    input('\nPress Enter to Continue')
    result_excluding_ncg4527 = hubble_constant_multiple_galaxies(ncg_distance, ncg_distance_error, False) # uses just the data from the other galaxies to get a

    print('\nHubble Constant Including ncg4527: ', round(result_including_ncg4527[0], 2), '+/-', round(result_including_ncg4527[1], 1)) # prints the final resu

    print('\nHubble Constant Excluding ncg4527: ', round(result_excluding_ncg4527[0], 2), '+/-', round(result_excluding_ncg4527[1], 1)) # prints the final resu

Main() # starts the code
```