Appendix

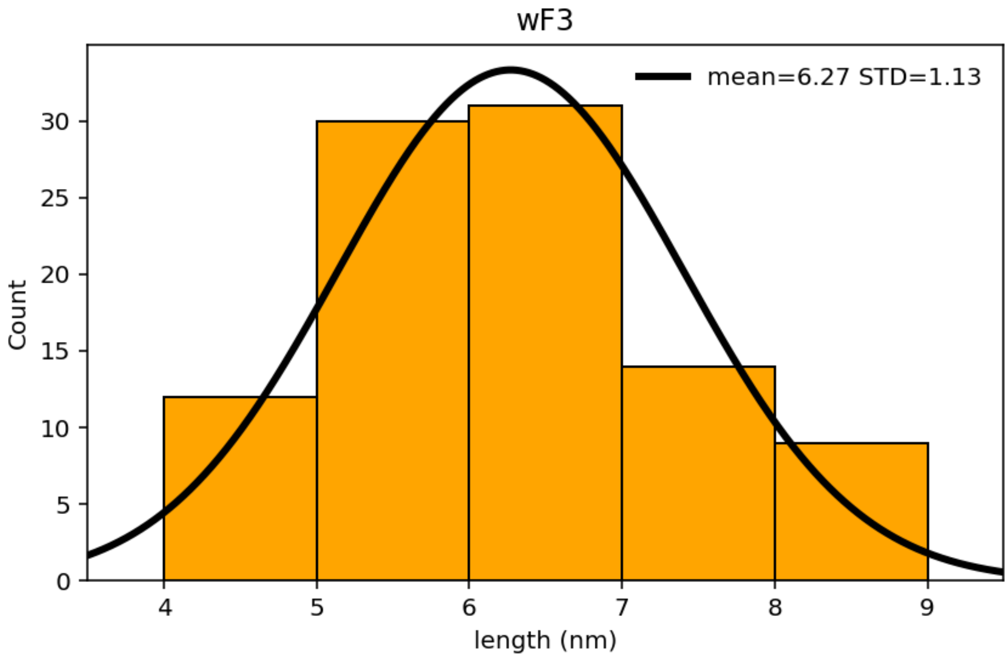


Figure 1 Thickness measurements for Ag nanoplates and prisms with mean and standard deviation

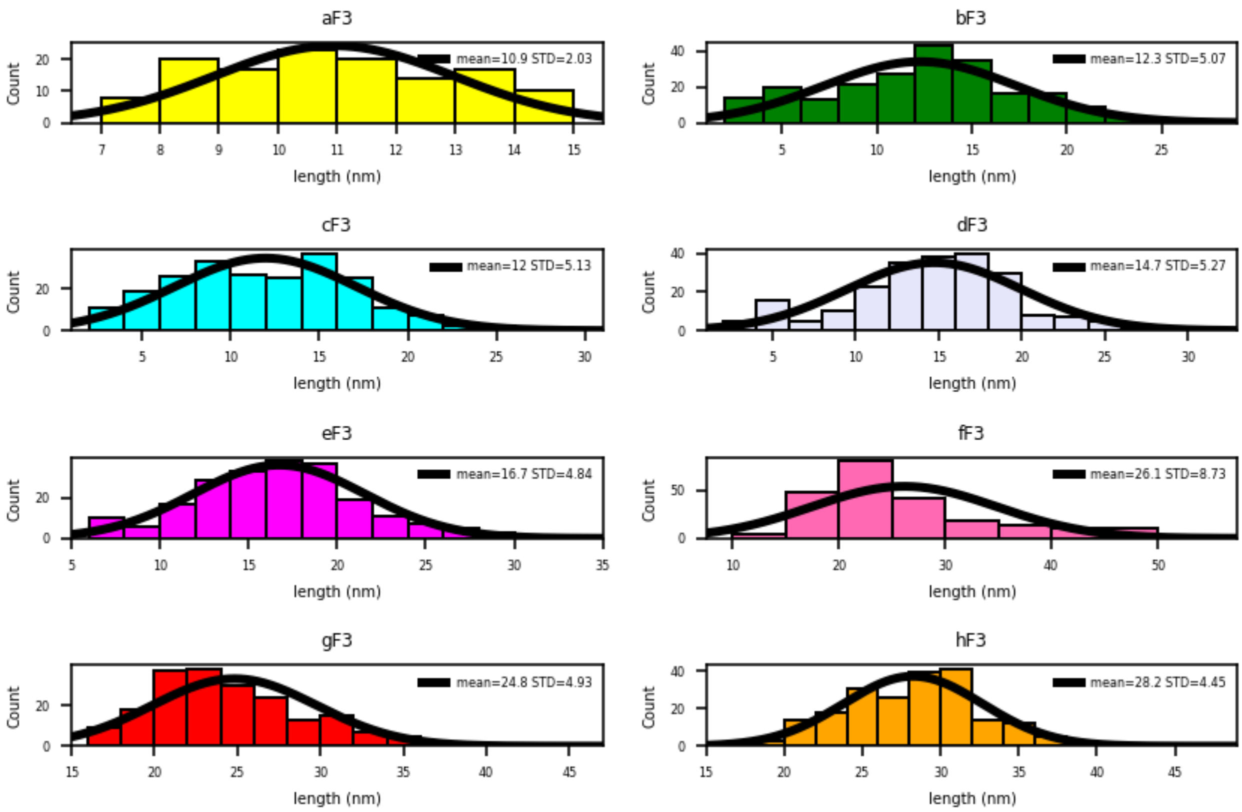


Figure 2 Length measurements for Ag nanoplates and prisms with mean and standard deviation

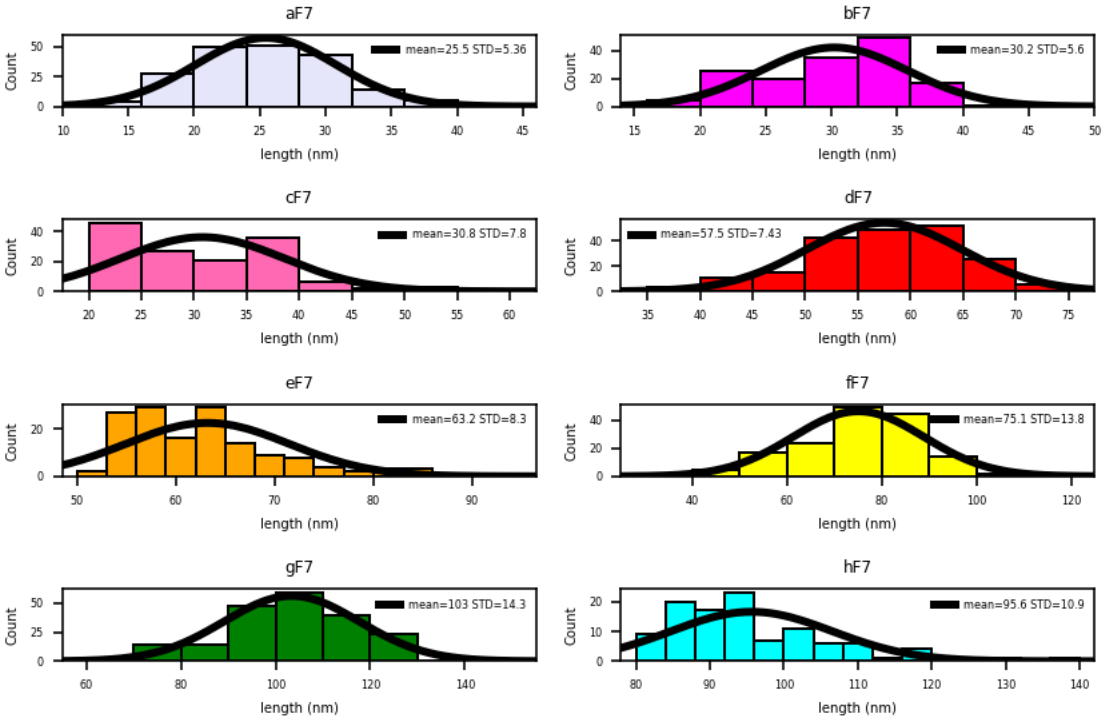


Figure 3 Length measurements for Au nanoplates and prisms with mean and standard deviation

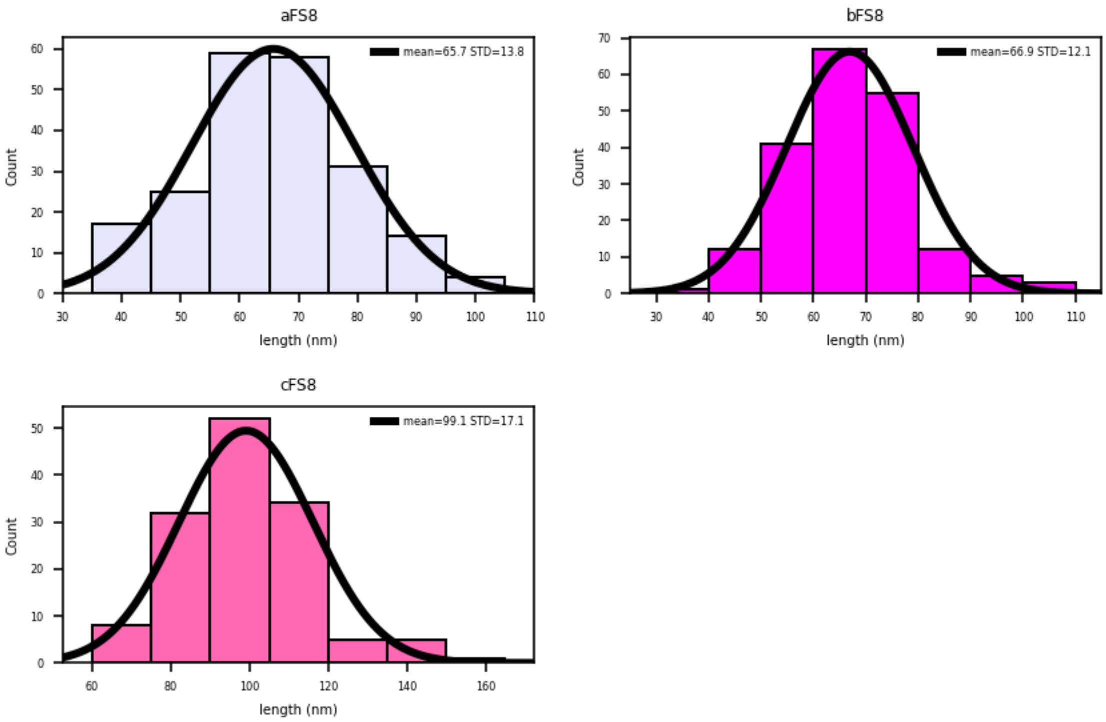


Figure 4 More length measurements for Au nanoplates and prisms with mean and standard deviation

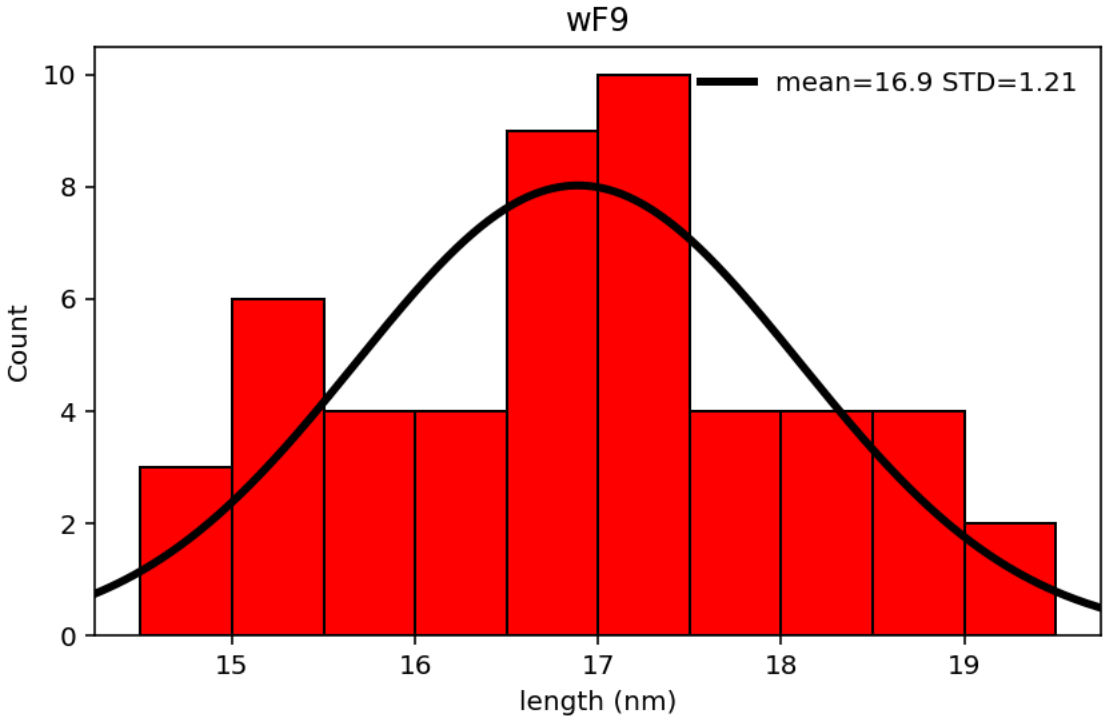


Figure 5 Thickness measurements for Au nanorods with mean and standard deviation

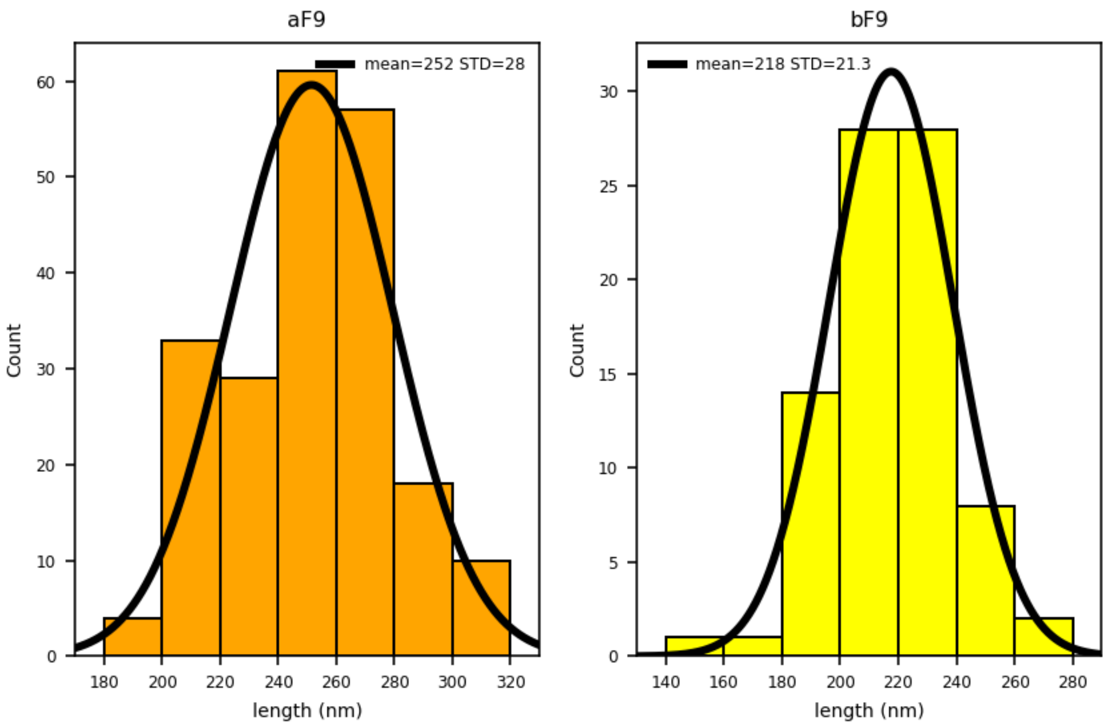


Figure 6 Length measurements for Au nanorods with mean and standard deviation

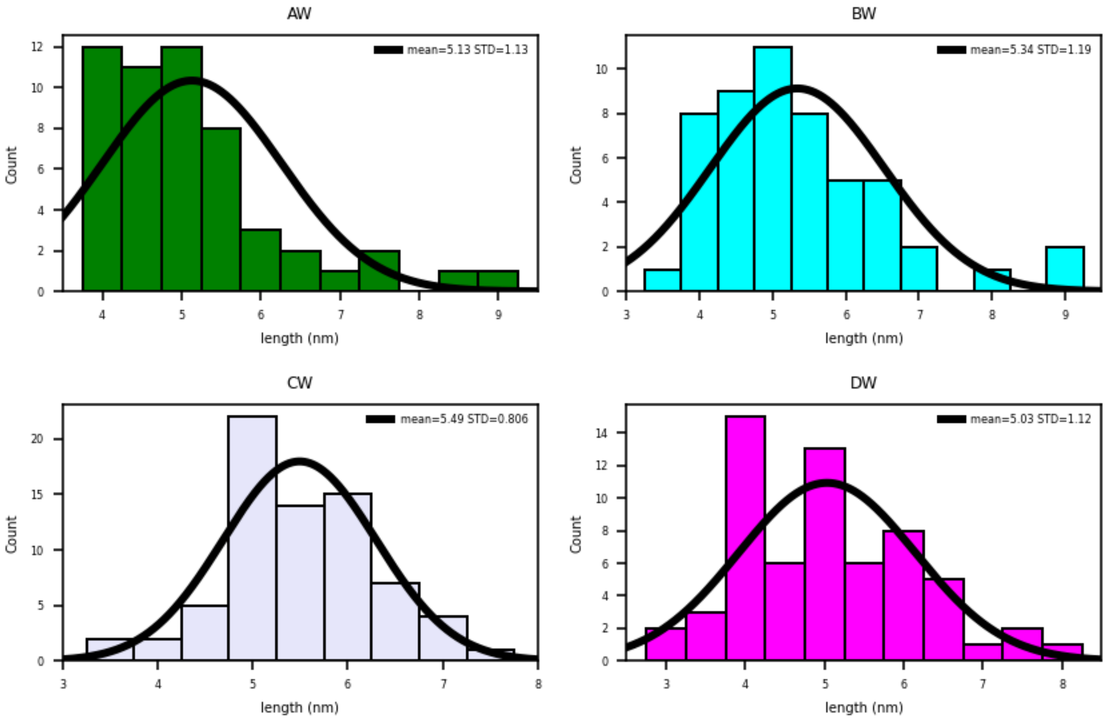


Figure 7 Thickness measurements for AgAu nanoplates and prisms with mean and standard deviation

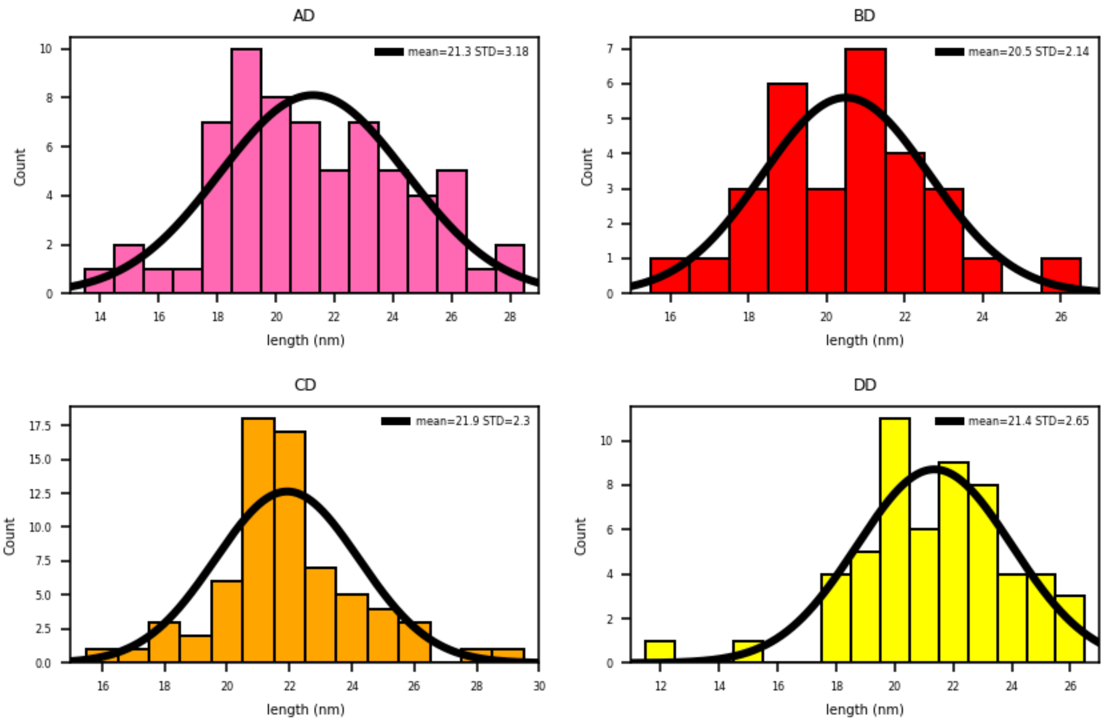
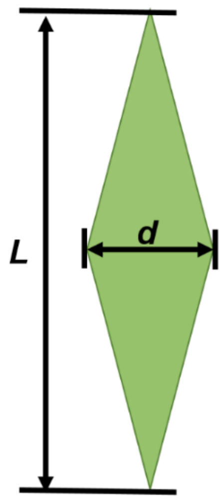
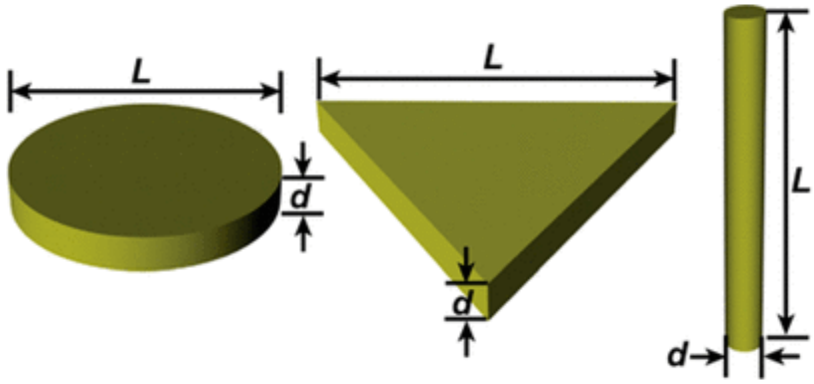


Figure 8 Length measurements for AgAu nanoplates and prisms with mean and standard deviation



[A1]

Figure 9 How aspect ratio was defined for nano bipyramids



[A1]

Figure 10 How aspect ratio was defined for nanoplates (left), nanoprisms (middle) and nanorods (right)

Error code and prepping the data

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from Group_project_starter_code import weight

data_1 = np.genfromtxt('errors.csv', delimiter=',',skip_header=1, dtype=str)
data_2 = np.genfromtxt('more errors.csv', delimiter=',',skip_header=1, dtype=str)
data = np.concatenate((data_1, data_2), axis=0) #combines the two sets of data
"""
List of dictionaries
"""
data_dict = {}
data_dict_2={}
data_dict_4={}
error_dict={}
mean_dict={}
perc_dict={}
aspect_ratio_dict={}
aspect_ratio_error_dict={}
weight_dict={}
std_ar_dict={}
perc_dict_std={}
max_dict={}
"""
functions
"""
def data_fixing(x):
    result = np.array([], dtype=float)  #creates an array of float data
    for row in x:
        count = int(row[0])  #first column is the count
        value = row[1]       #second column is the value
        result = np.append(result, value * np.ones(count))  #creates a full expanded array of data values
    return result

def product_error(e1,e2):
    return np.sqrt((e1**2)+(e2**2)) #returns the percentage error of the resultant mean

def update_dicts(aspect_key, key, thick_key, perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict, perc_dict_std,std_ar_dict):
    """
    Updates aspect ratio, aspect ratio error and standard deviation for aspect ratio dictionaries
    """
    mean_value = mean_dict[key] / mean_dict[thick_key] #mean resultant value
    error_value = product_error(perc_dict[key], perc_dict[thick_key])*mean_value #total standard error of the new mean
    std_value = product_error(perc_dict_std[key],perc_dict_std[thick_key])*mean_value #standard deviation of the new mean
    if aspect_key in aspect_ratio_dict:
        #if the key already exists add the new value
        aspect_ratio_error_dict[aspect_key] = np.vstack([aspect_ratio_error_dict[aspect_key], error_value])
        aspect_ratio_dict[aspect_key] = np.vstack([aspect_ratio_dict[aspect_key], mean_value])
        std_ar_dict[aspect_key]=np.vstack([std_ar_dict[aspect_key],std_value])
    else:
        #if the key doesnt exist create a new key with the value
        aspect_ratio_error_dict[aspect_key] = error_value
        aspect_ratio_dict[aspect_key] = mean_value
        std_ar_dict[aspect_key]=std_value
```

Figure 11 Loading in the data and initialising all the used dictionaries

```python
# Iterate through the array
for row in data:
    label = row[2]  #key name
    values = row[:2].astype(float)  # values asociated with that data set

    # If the data set already exists add the values to it
    if label in data_dict:
        data_dict[label] = np.vstack([data_dict[label], values])
    else:
        # Otherwise, create a new key with the first value
        data_dict[label] = values

for key in data_dict:
    #use the data fixing function to expand each data set out
    data_dict_2[key]=data_fixing(data_dict[key])
for key in data_dict_2:
    #find the standard error of each new data point
    error_dict[key]=np.std(data_dict_2[key])/np.sqrt(len(data_dict_2[key]))
for key in data_dict_2:
    #find the standard deviation of each new data point
    weight_dict[key]=np.std(data_dict_2[key])

for key in data_dict_2:
    #find the new data points for the new data set
    mean_dict[key]=np.mean(data_dict_2[key])
"""
the mean is taken from the report
the error and weight are assigned values so they dont return errors
"""
mean_dict['wF7']=7.5
error_dict['wF7']=1
weight_dict['wF7']=1

for key in mean_dict:
    #find the percentage standard error on each data point
    perc_dict[key]=error_dict[key]/mean_dict[key]
for key in mean_dict:
    #find the percentage standard deviation on each data point
    perc_dict_std[key]=weight_dict[key]/mean_dict[key]
"""
the percentage standard deviation of the width(thickness/short axis length) of gold
nanoplates/nanoprisms is assumed to be the same as that of silver nanoplates/nanoprisms,
percentage standard error on the gold nanoplates/nanoprisms is taken to be the standard deviation
"""
perc_dict['wF7']=perc_dict_std['wF3']
perc_dict_std['wF7']=perc_dict_std['wF3']
"""
define which keys are which nanoparticles
the letter before the F determines which graph it refers to the Fx determines which
figure its from a w before the F refers to width(thickeness/short axis length)
"""
ag_nanoplates=['aF3','bF3','cF3','dF3','eF3','fF3','gF3','hF3','iF3','jF3','kF3']
au_nanoplates=['aF7','bF7','cF7','dF7','eF7','fF7','gF7','hF7','aFS8','bFS8','cFS8']
```

Figure 12 sorting the data into respective dictionaries

```python
figure its from a w before the F refers to width(thickeness/short axis length)
"""
ag_nanoplates=['aF3','bF3','cF3','dF3','eF3','fF3','gF3','hF3','iF3','jF3','kF3']
au_nanoplates=['aF7','bF7','cF7','dF7','eF7','fF7','gF7','hF7','aFS8','bFS8','cFS8']
au_nanorods=['aF9','bF9']

"""
the last four elif statements refer to the data in data_2 from a different paper
they are seperated as they contain different amounts of gold plating
"""
# Iterate through keys in relevant dictionaries
for key in perc_dict:
    if key in ag_nanoplates:
        update_dicts('Ag_NP', key, 'wF3', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key in au_nanoplates:
        update_dicts('Au_NP', key, 'wF7', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key in au_nanorods:
        update_dicts('Au_NR', key, 'wF9', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key == 'AD':
        update_dicts('AuAg_NP', key, 'AW', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key == 'BD':
        update_dicts('AuAg_NP', key, 'BW', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key == 'CD':
        update_dicts('AuAg_NP', key, 'CW', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)
    elif key == 'DD':
        update_dicts('AuAg_NP', key, 'DW', perc_dict, mean_dict, aspect_ratio_dict, aspect_ratio_error_dict,perc_dict_std,std_ar_dict)

"""
loads in data taken from data sets where the standard deviation was already determined
"""
data_3=np.genfromtxt('data with errors.csv', delimiter=',',skip_header=1, dtype=str)

for row in data_3:
    #uses an almost identical loop as before to add to the relevant dictionaries
    label_1 = row[4]
    values_1 = row[1:3].astype(float)
    values_1_2=row[5].astype(float)

    if label_1 in aspect_ratio_dict:
        aspect_ratio_dict[label_1] = np.vstack([aspect_ratio_dict[label_1], values_1[0]])
        aspect_ratio_error_dict[label_1] = np.vstack([aspect_ratio_error_dict[label_1], values_1[1]/np.sqrt(values_1_2)])
        std_ar_dict[label_1] = np.vstack([std_ar_dict[label_1], values_1[1]])
    else:
        aspect_ratio_dict[label_1] = values_1[0]
        aspect_ratio_error_dict[label_1] = values_1[1]/np.sqrt(values_1_2)
        std_ar_dict[label_1]=values_1[1]

"""
loads in the relevant sensitivities and adds them to the relevant data points
"""
data_4=np.genfromtxt('this better work_2.csv', delimiter=',',skip_header=1, dtype=str)

#adds values to there own dictionary
for row in data_4:
    #once again uses the same loop
```

Figure 13 calculating the standard deviations and adding more data to the dictionaries

```python
        std_ar_dict[label_1] = np.vstack([std_ar_dict[label_1], values_1[1]])
    else:
        aspect_ratio_dict[label_1] = values_1[0]
        aspect_ratio_error_dict[label_1] = values_1[1]/np.sqrt(values_1_2)
        std_ar_dict[label_1]=values_1[1]

"""
loads in the relevant sensitivities and adds them to the relevant data points
"""
data_4=np.genfromtxt('this better work_2.csv', delimiter=',',skip_header=1, dtype=str)

#adds values to there own dictionary
for row in data_4:
    #once again uses the same loop
    label_2 = row[2]
    values_2 = row[0].astype(float)
    if label_2 in data_dict_4:
        data_dict_4[label_2] = np.vstack([data_dict_4[label_2], values_2])
    else:
        data_dict_4[label_2]  = values_2


for key in data_dict_4:
    #adds them to the relevant values in the aspect ratio dictionary
    aspect_ratio_dict[key]=np.hstack([aspect_ratio_dict[key],data_dict_4[key]])
for key in aspect_ratio_error_dict:
    #adds the standard errors to the relevant values in the aspect ratio dictionary
    aspect_ratio_dict[key]=np.hstack([aspect_ratio_dict[key],aspect_ratio_error_dict[key]])
"""
plots the bar charts of each data points and fits a gaussian to them doesnt use
"""

def gaussian(x, A):
    #defines the gaussian curve for curve fit
    return A * np.exp(-((x - mean_dict[key])**2) / (2 * weight_dict[key]**2))/(weight_dict[key]*np.sqrt(2*np.pi))
#list of colors we want to cylce through
colors = ['red', 'orange', 'yellow','green','cyan','#E6E6FA','magenta','#FF69B4']
count=0
for key in data_dict:
    #defines a count so it can cycle through colors
    if 30<=count<=33:
        plt.subplot(2,2,count-29)
        count=count+1
        plt.bar(data_dict[key][:,1],data_dict[key][:,0],width=data_dict[key][1,1]-data_dict[key][0,1],color=colors[count % len(colors)],edgecolor='black')
        plt.title(key,fontsize=6)
        x=np.linspace(2*data_dict[key][0,1]-data_dict[key][1,1],2*data_dict[key][-1,1]-data_dict[key][-2,1],10000)
        initial_guess = [max(data_dict[key][:,0])]
        #fits to find how to scale the gaussian to give the best looking graph
        popt, pcov = curve_fit(gaussian, data_dict[key][:,1], data_dict[key][:,0], p0=initial_guess)
        max_dict[key]=popt[0]
        plt.plot(x,gaussian(x,popt[0]),linestyle='-',c='Black',linewidth=3,label=f'mean={mean_dict[key]:.3g} STD={weight_dict[key]:.3g}')
        plt.ylabel('Count',fontsize=5)
        plt.xlabel('length (nm)',fontsize=5)

        plt.xlim(2*data_dict[key][0,1]-data_dict[key][1,1],2*data_dict[key][-1,1]-data_dict[key][-2,1])
        plt.tick_params(axis='y', labelsize=4)
        plt.tick_params(axis='x', labelsize=4)
        plt.legend(frameon=False,fontsize=4)
    else:
        count= count+1
plt.tight_layout()
plt.savefig('figure_extra data.png')
plt.show()
```

Figure 14 appending the respective sensitivities to their aspect ratio measurement and plotting the measurements taken of the length and with measurements

```python
import Group_project_starter_code as gps
import matplotlib.pyplot as plt
import math
import numpy as np
from scipy.optimize import curve_fit
from sklearn.linear_model import HuberRegressor
from scipy.stats import t
from matplotlib.lines import Line2D
import time
from Error_code import aspect_ratio_dict,std_ar_dict
import statsmodels.api as sm

#starts the timer
start_time = time.time()

#loads in the critical t values
data_t=np.loadtxt('critical t values.csv',skiprows=1,delimiter=',')

"""
defines the fitting function for the code
"""
def Fit(X,Y,W):
    #applies the functions defined in GPS
    FI=gps.F_info(W,X)
    L,U=gps.L_triag(FI)
    if gps.det_F_info_1(L) == 0:
        print("Matrix is singular, cannot compute inverse.")
    else:
        L_inv = gps.inverse_lower_triangular(L)
        U_inv = gps.inverse_upper_triangular(U)
        inv = gps.inverse(L_inv, U_inv)
        intercept = gps.a(inv, W, X, Y)
        gradient = gps.b(inv, W, X, Y)
        grad_err = gps.a_err(inv)
        inter_err=gps.b_err(inv)
    return gradient, intercept, grad_err,inter_err


def linear_model(x, a, b):
    return a * x + b

"""
creates the dictionaries to be used in the code
"""
grad_and_int={}
grad_and_int_OLS={}
grad_and_int_hub={}
x=0
"""
As sensitivity errors are assumed to be minimal compared to aspect ratio errors the code fits a line
with aspect ratio on the y axis and sensitivity on the x then reverses the found gradients it does this
to give ease of comparrison between similar graphs in other reports
"""
for key in aspect_ratio_dict:
    grad_and_int[key]=Fit(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],1/(std_ar_dict[key][:,0]**2))
    grad_and_int_OLS[key]=Fit(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],np.ones(np.shape(aspect_ratio_dict[key][:,0])))
    X_reshape=aspect_ratio_dict[key][:,1].reshape(-1,1)
    huber=HuberRegressor()
    huber.fit(X_reshape,aspect_ratio_dict[key][:,0],sample_weight=(1/std_ar_dict[key][:,0]**2))
    grad_and_int_hub[key]=np.array([1/huber.coef_[0],-huber.intercept_/huber.coef_[0]])
    if x==0:
```

Figure 15 exporting the data into a new script and calculating fits for the gradient and intercept of each shape individually

```python
for key in aspect_ratio_dict:
    grad_and_int[key]=Fit(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],1/(std_ar_dict[key][:,0]**2))
    grad_and_int_OLS[key]=Fit(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],np.ones(np.shape(aspect_ratio_dict[key][:,0])))
    X_reshape=aspect_ratio_dict[key][:,1].reshape(-1,1)
    huber=HuberRegressor()
    huber.fit(X_reshape,aspect_ratio_dict[key][:,0],sample_weight=(1/std_ar_dict[key][:,0]**2))
    grad_and_int_hub[key]=np.array([1/huber.coef_[0],-huber.intercept_/huber.coef_[0]])
    if x==0:
        #initialises the array Com(all nanoparticles sensitivity and aspect ratio), ERR(all nanoparticle standard errors) and w(all nanoparticles standard deviation)
        Com=aspect_ratio_dict[key][:,:2]
        Err=aspect_ratio_dict[key][:,2]
        w=std_ar_dict[key][:,0]
        x=x+1
    else:
        #combines all the values into the new arrays
        Com=np.vstack([Com,aspect_ratio_dict[key][:,:2]])
        Err=np.concatenate([Err,aspect_ratio_dict[key][:,2]])
        w=np.concatenate([w,std_ar_dict[key][:,0]])

#gives the colors in the same way as in Error_code but does shapes aswell
colours=['#FF2400','#4169E1','#66B200','#FF7F00','#FF69B4']
shapes=['o','o','X','o','^']
count=0



for key in aspect_ratio_dict:
    if key=='Au_NR':
        #if the key is either AuAg_NP or Au_NR dont plot the predicted curves
        #plot the standard errorbars and the standard deviation show each graph individually
        #plt.subplot(2,2,count+1)
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key}',markersize=6,capsize=2,ecolor='black',markeredgecolor='black')
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=aspect_ratio_dict[key][:,2][:],marker='none',linestyle='none',capsize=2,ecolor='red')
        #plt.title(f'{key} Sensitivity against Aspect Ratio')
        #plt.xlabel('Aspect Ratio')
        #plt.ylabel('Sensitivity (nm/RIU)')
        #plt.legend(loc='upper left',frameon=False)
        #plt.savefig(f'{key}.png')
        #plt.show()
        count=count+1
    elif key=='AuAg_NP':
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key}',markersize=6,capsize=2,ecolor='black',markeredgecolor='black')
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=aspect_ratio_dict[key][:,2][:],marker='none',linestyle='none',capsize=2,ecolor='red')

        count=count+1

    else:
        #otherwise plot the predicted curves
        #plot the standard errors and standard deviations
        #print the chisquared values of the OLS and WLS
        #plt.subplot(2,2,count+1)
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key}',markersize=6,capsize=2,ecolor='black',markeredgecolor='black')
        #plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key} $R^2$={gps.R_squared(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],1/grad_a
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=aspect_ratio_dict[key][:,2][:],marker='none',linestyle='none',capsize=2,ecolor='red')
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),(1/grad_and_int[key][0])*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)-(grad_and_int[key][1]/grad_and_int[
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),(1/grad_and_int_OLS[key][0])*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)-grad_and_int_OLS[key][1]/grad_an
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),grad_and_int_hub[key][0]*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)+grad_and_int_hub[key][1],c=colours[
        #plt.title(f'{key} Sensitivity against Aspect Ratio')
        #plt.xlabel('Aspect Ratio')
        #plt.ylabel('Sensitivity (nm/RIU)')
        #plt.legend(loc='upper left',frameon=False)
        #plt.savefig(f'{key}.png')
        #plt.show()
```

Figure 16 plotting all the fitted curves including external modules and combining data

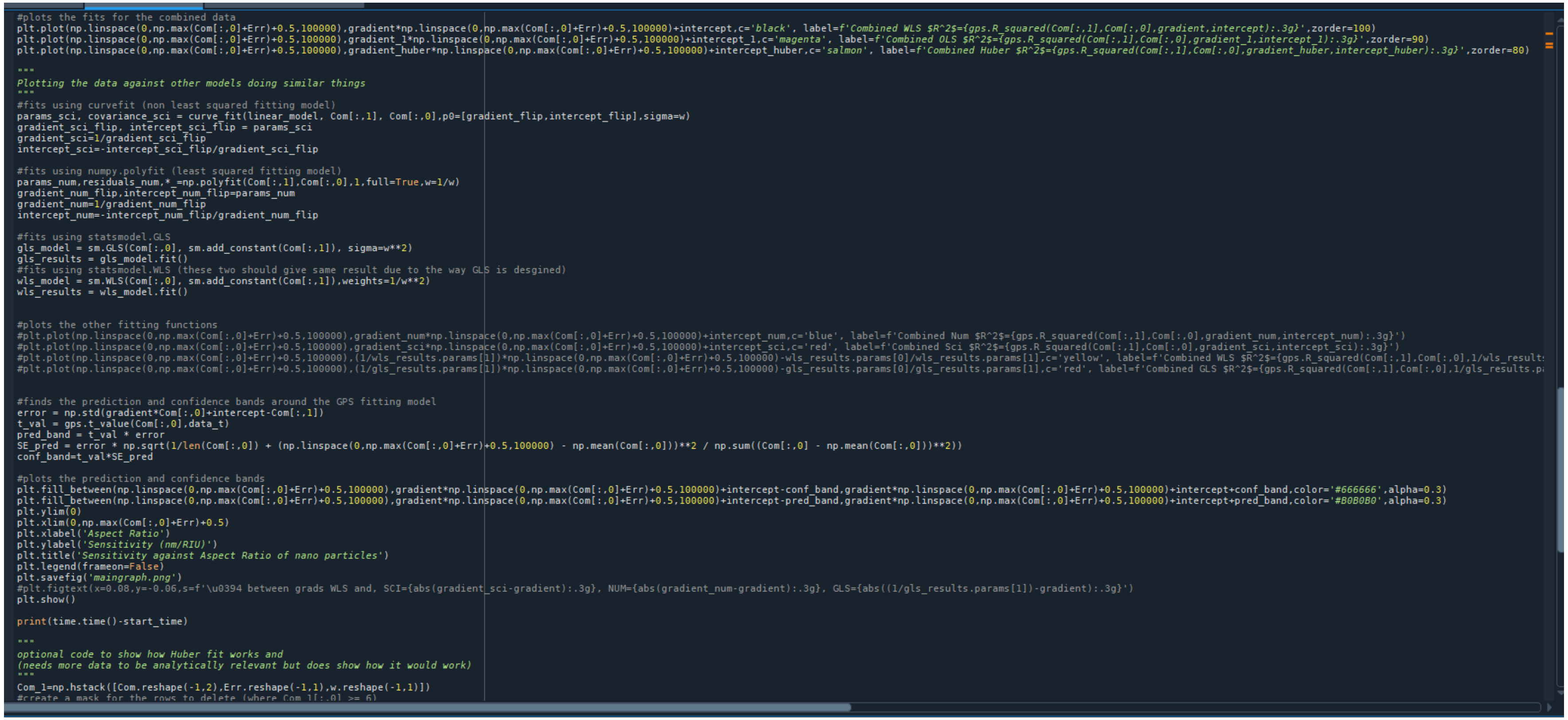
```python
    else:
        #otherwise plot the predicted curves
        #plot the standard errors and standard deviations
        #print the chisquared values of the OLS and WLS
        #plt.subplot(2,2,count+1)
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key}',markersize=6,capsize=2,ecolor='black',markeredgecolor='black')
        #plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=std_ar_dict[key][:,0],marker=shapes[count],c=colours[count],linestyle='none',label=f'{key} $R^2$={gps.R_squared(aspect_ratio_dict[key][:,1],aspect_ratio_dict[key][:,0],1/grad_
        plt.errorbar(aspect_ratio_dict[key][:,0],aspect_ratio_dict[key][:,1],xerr=aspect_ratio_dict[key][:,2][:],marker='none',linestyle='none',capsize=2,ecolor='red')
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),(1/grad_and_int[key][0])*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)-(grad_and_int[key][1]/grad_and_int[
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),(1/grad_and_int_OLS[key][0])*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)-grad_and_int_OLS[key][1]/grad_an
        #plt.plot(np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000),grad_and_int_hub[key][0]*np.linspace(0,np.max(aspect_ratio_dict[key][:,0]+aspect_ratio_dict[key][:,2])+0.5,100000)+grad_and_int_hub[key][1],c=colours[
        #plt.title(f'{key} Sensitivity against Aspect Ratio')
        #plt.xlabel('Aspect Ratio')
        #plt.ylabel('Sensitivity (nm/RIU)')
        #plt.legend(loc='upper left',frameon=False)
        #plt.savefig(f'{key}.png')
        #plt.show()
        #print(key)
        #print(gps.Chi_squared_red(aspect_ratio_dict[key][:,0], aspect_ratio_dict[key][:,1], grad_and_int_OLS[key][0], grad_and_int_OLS[key][1], std_ar_dict[key]))
        #print(gps.Chi_squared_red(aspect_ratio_dict[key][:,0], aspect_ratio_dict[key][:,1], grad_and_int[key][0], grad_and_int[key][1], std_ar_dict[key]))
        count=count+1


#fits in the same way as before OLS, WLS and Huber to the data
gradient_flip,intercept_flip,grad_err_flip,inter_err_flip=Fit(Com[:,1],Com[:,0],1/w**2)
gradient_1_flip,intercept_1_flip,grad_1_err_flip,inter_1_err_flip=Fit(Com[:,1],Com[:,0],np.ones(np.shape(Com[:,1])))
gradient=1/gradient_flip
intercept=-intercept_flip/gradient_flip
gradient_1=1/gradient_1_flip
intercept_1=-intercept_1_flip/gradient_1_flip
X_reshape=Com[:,1].reshape(-1,1)
huber = HuberRegressor()
huber.fit(X_reshape, Com[:,0],sample_weight=(1/w**2))
gradient_huber = 1/huber.coef_[0]
intercept_huber = -huber.intercept_/huber.coef_[0]
#plots the fits for the combined data
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept,c='black', label=f'Combined WLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient,intercept):.3g}',zorder=100)
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_1*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_1,c='magenta', label=f'Combined OLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_1,intercept_1):.3g}',zorder=90)
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_huber*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_huber,c='salmon', label=f'Combined Huber $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_huber,intercept_huber):.3g}',zorder=80)

"""
Plotting the data against other models doing similar things
"""
#fits using curvefit (non least squared fitting model)
params_sci, covariance_sci = curve_fit(linear_model, Com[:,1], Com[:,0],p0=[gradient_flip,intercept_flip],sigma=w)
gradient_sci_flip, intercept_sci_flip = params_sci
gradient_sci=1/gradient_sci_flip
intercept_sci=-intercept_sci_flip/gradient_sci_flip

#fits using numpy.polyfit (least squared fitting model)
params_num,residuals_num,*_=np.polyfit(Com[:,1],Com[:,0],1,full=True,w=1/w)
gradient_num_flip,intercept_num_flip=params_num
gradient_num=1/gradient_num_flip
```

Figure 17 more plotting and statistical analysis



```
#plots the fits for the combined data
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept,c='black', label=f'Combined WLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient,intercept):.3g}',zorder=100)
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_1*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_1,c='magenta', label=f'Combined OLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_1,intercept_1):.3g}',zorder=90)
plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_huber*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_huber,c='salmon', label=f'Combined Huber $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_huber,intercept_huber):.3g}',zorder=80)

"""
Plotting the data against other models doing similar things
"""
#fits using curvefit (non least squared fitting model)
params_sci, covariance_sci = curve_fit(linear_model, Com[:,1], Com[:,0],p0=[gradient_flip,intercept_flip],sigma=w)
gradient_sci_flip, intercept_sci_flip = params_sci
gradient_sci=1/gradient_sci_flip
intercept_sci=-intercept_sci_flip/gradient_sci_flip

#fits using numpy.polyfit (least squared fitting model)
params_num,residuals_num,*_=np.polyfit(Com[:,1],Com[:,0],1,full=True,w=1/w)
gradient_num_flip,intercept_num_flip=params_num
gradient_num=1/gradient_num_flip
intercept_num=-intercept_num_flip/gradient_num_flip

#fits using statsmodel.GLS
gls_model = sm.GLS(Com[:,0], sm.add_constant(Com[:,1]), sigma=w**2)
gls_results = gls_model.fit()
#fits using statsmodel.WLS (these two should give same result due to the way GLS is desgined)
wls_model = sm.WLS(Com[:,0], sm.add_constant(Com[:,1]),weights=1/w**2)
wls_results = wls_model.fit()


#plots the other fitting functions
#plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_num*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_num,c='blue', label=f'Combined Num $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_num,intercept_num):.3g}')
#plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient_sci*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept_sci,c='red', label=f'Combined Sci $R^2$={gps.R_squared(Com[:,1],Com[:,0],gradient_sci,intercept_sci):.3g}')
#plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),(1/wls_results.params[1])*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)-wls_results.params[0]/wls_results.params[1],c='yellow', label=f'Combined WLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],1/wls_result
#plt.plot(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),(1/gls_results.params[1])*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)-gls_results.params[0]/gls_results.params[1],c='red', label=f'Combined GLS $R^2$={gps.R_squared(Com[:,1],Com[:,0],1/gls_results.pa


#finds the prediction and confidence bands around the GPS fitting model
error = np.std(gradient*Com[:,0]+intercept-Com[:,1])
t_val = gps.t_value(Com[:,0],data_t)
pred_band = t_val * error
SE_pred = error * np.sqrt(1/len(Com[:,0]) + (np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000) - np.mean(Com[:,0]))**2 / np.sum((Com[:,0] - np.mean(Com[:,0]))**2))
conf_band=t_val*SE_pred

#plots the prediction and confidence bands
plt.fill_between(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept-conf_band,gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept+conf_band,color='#666666',alpha=0.3)
plt.fill_between(np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000),gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept-pred_band,gradient*np.linspace(0,np.max(Com[:,0]+Err)+0.5,100000)+intercept+pred_band,color='#B0B0B0',alpha=0.3)
plt.ylim(0)
plt.xlim(0,np.max(Com[:,0]+Err)+0.5)
plt.xlabel('Aspect Ratio')
plt.ylabel('Sensitivity (nm/RIU)')
plt.title('Sensitivity against Aspect Ratio of nano particles')
plt.legend(frameon=False)
plt.savefig('maingraph.png')
#plt.figtext(x=0.08,y=-0.06,s=f'\u0394 between grads WLS and, SCI={abs(gradient_sci-gradient):.3g}, NUM={abs(gradient_num-gradient):.3g}, GLS={abs((1/gls_results.params[1])-gradient):.3g}')
plt.show()

print(time.time()-start_time)

"""
optional code to show how Huber fit works and
(needs more data to be analytically relevant but does show how it would work)
"""
Com_1=np.hstack([Com.reshape(-1,2),Err.reshape(-1,1),w.reshape(-1,1)])
#create a mask for the rows to delete (where Com_1[:,0] >= 6)
```

Figure 18 more plotting and analysis

```python
print(time.time()-start_time)

"""
optional code to show how Huber fit works and
(needs more data to be analytically relevant but does show how it would work)
"""
Com_1=np.hstack([Com.reshape(-1,2),Err.reshape(-1,1),w.reshape(-1,1)])
#create a mask for the rows to delete (where Com_1[:,0] >= 6)
mask = Com_1[:, 0] < 6

#apply the mask to filter out the rows
Com_1_filtered = Com_1[mask]

#plot the scatter plot of Com_1 after filtering
#plt.errorbar(Com_1_filtered[:, 0], Com_1_filtered[:, 1],Com_1_filtered[:,3],marker='o',linestyle='none',capsize=10)

#fit to the filtered data
grad_2, int_2, grad_err_2, int_err_2 = Fit(Com_1_filtered[:, 1], Com_1_filtered[:, 0], 1 / Com_1_filtered[:, 3]**2)

#plot OLS and Huber of original data and the fit of the new data
#plt.plot(np.linspace(0,np.max(Com_1_filtered[:,0])+0.5,100000),gradient_huber*np.linspace(0,np.max(Com_1_filtered[:,0])+0.5,100000)+intercept_huber,c='salmon', label=f'Combined Huber COM $R^2$={gps.R_squared(Com_1_filtered[:,1],Com_1_filtered[:,0],gradient_
#plt.plot(np.linspace(0,np.max(Com_1_filtered[:,0])+0.5,100000),1/grad_2*np.linspace(0,np.max(Com_1_filtered[:,0])+0.5,100000)-int_2/grad_2,c='purple', label=f'Combined WLS COM FILT $R^2$={gps.R_squared(Com_1_filtered[:,1],Com_1_filtered[:,0],1/grad_2,-int_2,
"""
compare the parameters of the new fit to the huber regression of the orignal to show that if
more data was obtained that the huber regressor would remove the outliers close to the limit
of linearity
"""

#plt.legend(frameon=False)
#plt.show()

#def Fit_2(X,Y,m,c,p):
    #res=gps.residuals(m, X, c, Y).reshape(-1,1)
    #res_T=res.T
    #cov_est=((res@res_T)/(len(X)-p))
    #cov_inv = np.linalg.inv(cov_est)
    #X_Des=np.hstack([np.ones(np.shape(X.reshape(-1,1))),X.reshape(-1,1)])
    #FI=gps.F_info_2(cov_inv, X_Des)
    #inv = np.linalg.inv(FI)
    #z=gps.matrix_problem(inv, X_Des, cov_inv, Y.reshape(-1,1))
    #intercept=z[0]
    #gradient=z[1]
    #return gradient,intercept
#gradient_GLS_Flip,intercept_GLS_Flip=Fit_2(Com[:,1],Com[:,0], gradient_flip, intercept_flip, 2)
#grad_1_GLS=gradient_GLS_Flip
#int_1_GLS=intercept_GLS_Flip
#grad_2_GLS=gradient_flip
#int_2_GLS=intercept_flip
#n=0
```

Figure 19 creating a mask to filter data to show/highlight how Huber regression works

```
    #return gradient,intercept
#gradient_GLS_Flip,intercept_GLS_Flip=Fit_2(Com[:,1],Com[:,0], gradient_flip, intercept_flip, 2)
#grad_1_GLS=gradient_GLS_Flip
#int_1_GLS=intercept_GLS_Flip
#grad_2_GLS=gradient_flip
#int_2_GLS=intercept_flip
#n=0

#while (n < 100 and (abs(grad_1_GLS - grad_2_GLS) > 10**-10 or abs(int_1_GLS - int_2_GLS) > 10**-10)) or n < 10:
    #grad_2_GLS = grad_1_GLS
    #int_2_GLS = int_1_GLS
    #grad_1_GLS, int_1_GLS = Fit_2(Com[:, 1], Com[:, 0], grad_1_GLS, int_1_GLS, 2)
    #n=n+1
#print(n)
#print(gradient_flip-grad_1_GLS)
```

Figure 20 an attempt at estimating the covariance matrix that ultimately failed

```
"""
Weighted Least Squares Fit (matrix form)

The model will attempt to find the graident and intercept of a linear curve
"""
import numpy as np

"""
The weight function returns a scalar that is equal to one over the variance
of the 1 dimensional array inputted
"""

def weight(x):
    n = len(x)
    return 1 / ((np.dot(x, x) - (np.sum(x)**2) / n) / (n-1))

"""
F_info manually formulates the matrix (X^T @ W @ X) for a linear 1 variable
dependent Weighted Least Squares fit
"""
def F_info(W,X):
    c=np.dot(W,X)
    return np.array([[np.sum(W),c]
                     ,[c,np.dot(W,X*X)]])
def F_info_2(cov_inv,X):
    return X.T@(cov_inv@X)

"""
L_Triag computes the elements of L and then L^T which make up the lower and upper
matricies of the cholesky decomposition of the inputted matrix
"""

def L_triag(FI):
    #calculates the cholesky decomposition A=L^T @ L
    n = FI.shape[0]
    L = np.zeros((n, n))
    for i in range(n):
        for j in range(i+1):  #j goes from 0 to i (lower triangular part)
            if i == j:
                #diagonal elements of the lower triagonal
                sum_of_squares = np.sum(L[i, :i] ** 2)  #sum of squares of previous row elements
                L[i, i] = np.sqrt(FI[i, i] - sum_of_squares)
            else:
                #off diagonal elements of the lower triagonal
                sum_of_products = np.sum(L[i, :j] * L[j, :j])  #sum of product of previous elements
                L[i, j] = (FI[i, j] - sum_of_products) / L[j, j]
    L_T=L.T #transposing to get the upper triagonal
    return L, L_T



"""
As a result of cramers rule we can say that det(FI) of a cholesky decomposition
is det(L)*det(L_T) therefore det(FI)=product of diagonal of L
"""
```

Figure 21 defining the functions for future and current fitting for WLS and GLS

```python
"""
As a result of cramers rule we can say that det(FI) of a cholesky decomposition
is det(L)*det(L_T) therefore det(FI)=product of diagonal of L
"""


def det_F_info_1(L):
    det_F_I = np.prod(np.diag(L)) ** 2  #square of the product of the diagonal elements of L
    return det_F_I

"""
Using forwards substitution we can get the lower triagonal inverse of the
Matrix
"""

def inverse_lower_triangular(L):
    n = L.shape[0]
    L_inv = np.zeros_like(L)

    #loop over rows to calculate diagonal elements
    for i in range(n):
        L_inv[i, i] = 1 / L[i, i]

        #loop to calculate off-diagonal elements in the lower triangular part
        for j in range(i):
            sum_ = 0
            for k in range(j, i):
                sum_ += L[i, k] * L_inv[k, j]
            L_inv[i, j] = -sum_ / L[i, i]

    return L_inv
"""
Using backwards substitution we can get the lower triagonal inverse of the
Matrix
"""

def inverse_upper_triangular(U):
    n = U.shape[0]
    U_inv = np.zeros_like(U)

    #loop over rows in reverse order for backward substitution
    for i in reversed(range(n)):
        # Calculate the diagonal element
        U_inv[i, i] = 1 / U[i, i]

        #loop to calculate off-diagonal elements in the upper triangular part
        for j in range(i + 1, n):
            sum_ = 0
            for k in range(i + 1, j + 1):
                sum_ += U[i, k] * U_inv[k, j]
            U_inv[i, j] = -sum_ / U[i, i]

    return U_inv
"""
Given the rule A=L @ L^T and A^-1 = L^T^-1 @ L^-1 the code can compute A^-1 as long as
the cholesky decomposition conditions are met
```

Figure 22 checks and inversion of the triagonal matrices

```python
    return U_inv
"""
Given the rule A=L @ L^T and A^-1 = L^T^-1 @ L^-1 the code can compute A^-1 as long as
the cholesky decomposition conditions are met
"""

def inverse(L_inv,U_inv):
    #recombines the decomposed matrix
    return U_inv@L_inv

def matrix_problem(inv,X,cov_inv,Y):
    return inv@X.T@cov_inv@Y
"""
the matrix problem is formulated manually
"""
def a(inv,W,X,Y):
    return inv[0,0]*np.dot(Y,W)+inv[0,1]*np.dot(W,Y*X)
def b(inv,W,X,Y):
    return inv[1,0]*np.dot(Y,W)+inv[1,1]*np.dot(W,Y*X)
def a_err(inv):
    return np.sqrt(inv[0,0])
def b_err(inv):
    return np.sqrt(inv[1,1])
def residuals(m,X,c,Y):
    return Y-(m*X+c)



"""
statistical tests to measure how good a model is
"""

def R_squared(Y,X,m,c):
    Y_pred=m * X + c
    ss_total=np.sum((Y-np.mean(Y))**2)
    ss_residual=np.sum((Y-Y_pred)**2)
    r2=1-(ss_residual/ss_total)
    return r2

def Chi_squared_red(Y,X,m,c,sigma):
    Y_pred=m*X+c
    Chi=np.sum(((Y-Y_pred)**2)/sigma**2)
    return Chi/(len(X)-2)

def Mean_square_error(Y,X,m,c):
    Y_pred=m*X+c
    MSE=np.sum((Y-Y_pred)**2)/len(Y)
    return MSE

def t_value(X,d):
    z=0
    for x in d[:,0]:
        if (len(X)-2)-x>=0: #finds the appropriate t value index
            z=z+1
```

Figure 23 formulating the matrix problem and defining statistical tests

```python
def t_value(X,d):
    z=0
    for x in d[:,0]:
        if (len(X)-2)-x>=0: #finds the appropriate t value index
            z=z+1
        else:
            break
    return d[z,1]
```

[A2]

Figure 24 finding the critical t value from a list of pre-determined critical t values

[A1] Khan AU, Zhao S, Liu G. Key parameter controlling the sensitivity of plasmonic metal nanoparticles: aspect ratio. The Journal of Physical Chemistry C. 2016 Sep 1;120(34):19353-64.

[A2] Microsoft Word - Utts-Heckard_Ttable.doc