In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

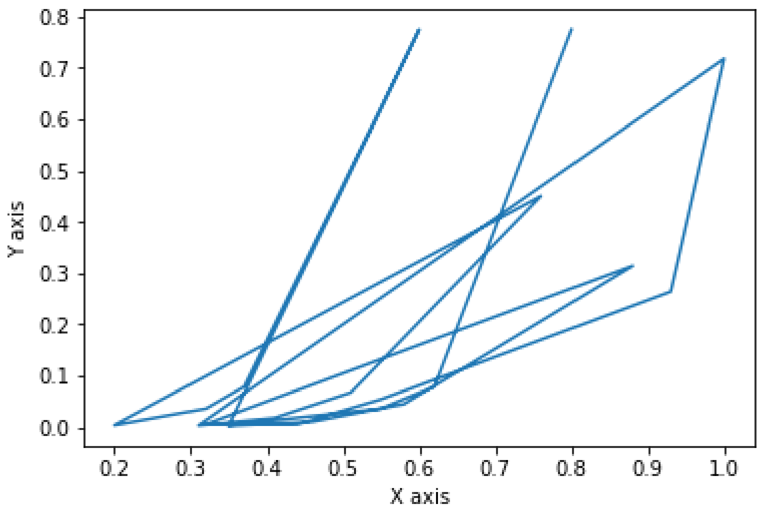
In [3]:
```
data1 = pd.read_csv('D:/class/Project/ML/sensitivity analysis/datset-1.csv')
data2 = pd.read_csv('D:/class/Project/ML/sensitivity analysis/ref dataset.csv')
```

In [4]:
```
data1.columns
```

Out[4]:
```
Index(['coat', 'time', 'in-vitro', 'in-vivo'], dtype='object')
```

In [5]:
```
#graph for dataset 1
```

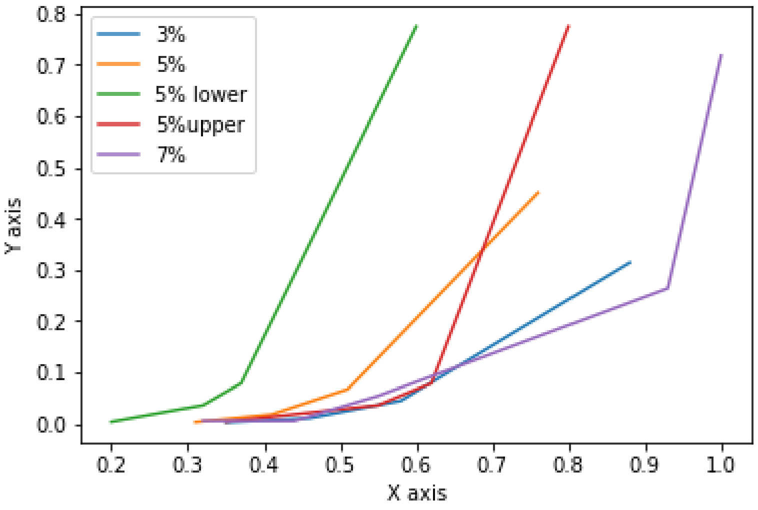
In [7]:
```
import matplotlib.pyplot as plt
x = data1['in-vitro']
y = data1['in-vivo']
plt.plot(x,y)
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



In [7]:
```
#graph to seperate every coat with x and y
```

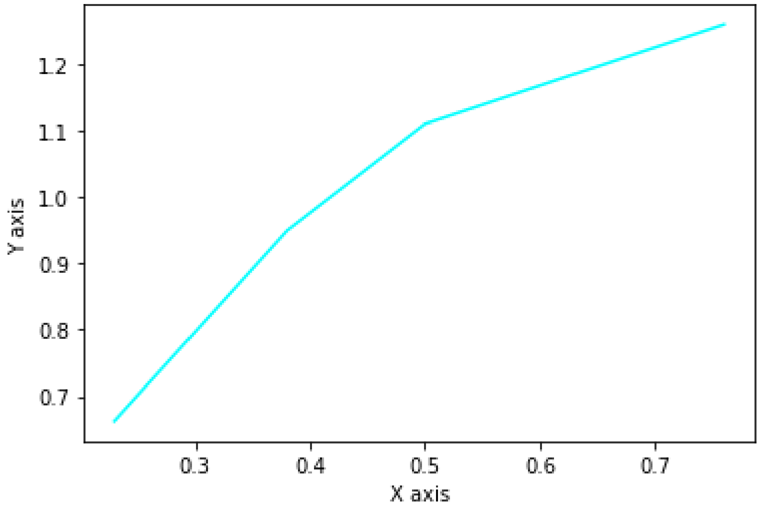
In [8]:
```
import pandas as pd
import matplotlib.pyplot as plt

# create a dataframe with x, y, and coat columns
x = data1['in-vitro']
y = data1['in-vivo']
coat1 = data1['coat']
data_dict = {'x': x, 'y': y, 'coat': coat1}
data = pd.DataFrame(data_dict)
print(data)
# group the data by coat type
grouped_data = data.groupby('coat')

# plot each group separately
for coat_type, group in grouped_data:
    plt.plot(group['x'], group['y'], label=coat_type)

#set x and y labels
plt.xlabel('X axis')
plt.ylabel('Y axis')

# add a legend
plt.legend()

# show the graph
plt.show()
```

	x	y	coat
0	0.35	0.002856	3%
1	0.46	0.011032	3%
2	0.58	0.044717	3%
3	0.88	0.314290	3%
4	0.32	0.006000	7%
5	0.44	0.005723	7%
6	0.55	0.054019	7%
7	0.93	0.264000	7%
8	1.00	0.718000	7%
9	0.31	0.003526	5%
10	0.41	0.017983	5%
11	0.51	0.067054	5%
12	0.76	0.450705	5%
13	0.20	0.004000	5% lower
14	0.32	0.036000	5% lower
15	0.37	0.079000	5% lower
16	0.60	0.775000	5% lower
17	0.35	0.004000	5%upper
18	0.55	0.036000	5%upper
19	0.62	0.079000	5%upper
20	0.80	0.775000	5%upper



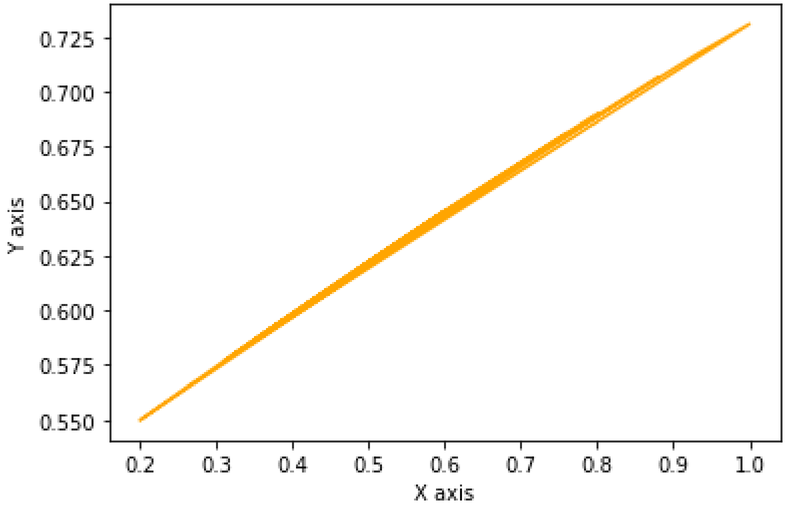
In [9]:
```
#graph for dataset-2
```

In [9]:
```
import matplotlib.pyplot as plt
x = data2['in-vitro']
y = data2['in-vivo']
plt.plot(x,y, color='cyan')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



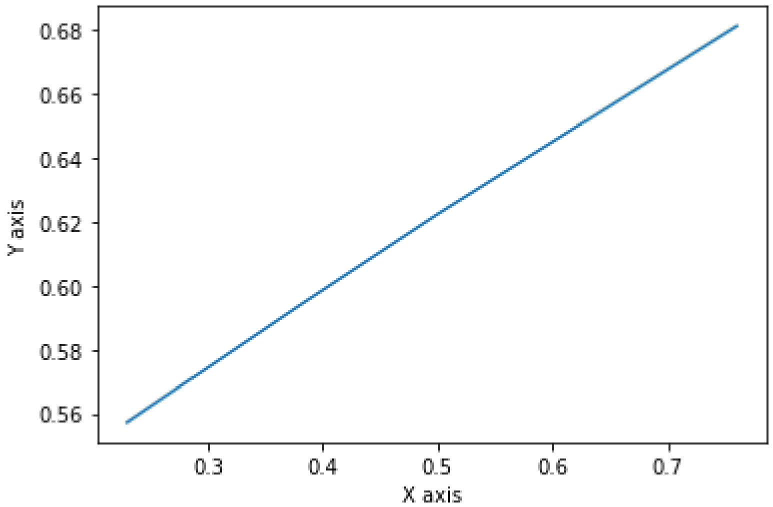
In [11]:
```
#sigmoid grapgh for dataset-1 using in-vitro
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

# define the sigmoid function
x = data1['in-vitro']
def sigmoid_func(x, a, b):
    return 1 / (1 + np.exp(-(x-a)/b))

# calculate y values using the sigmoid function
y = sigmoid_func(x, 0, 1)

# plot the graph
plt.plot(x, y, color = "orange")

# set x and y labels
plt.xlabel('X axis')
plt.ylabel('Y axis')

# show the graph
plt.show()
```



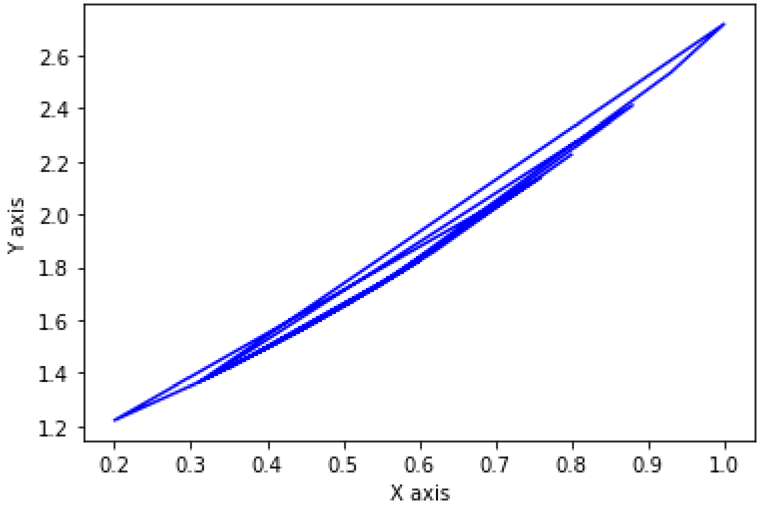
In [13]:
```
#sigmoid graph for dataset-1 using in-vitro
```

In [11]:
```
import numpy as np
import matplotlib.pyplot as plt

# define the sigmoid function
x = data2['in-vitro']
def sigmoid_func(x, a, b):
    return 1 / (1 + np.exp(-(x-a)/b))

# calculate y values using the sigmoid function
y = sigmoid_func(x, 0, 1)

# plot the graph
plt.plot(x, y)

# set x and y labels
plt.xlabel('X axis')
plt.ylabel('Y axis')

# show the graph
plt.show()
```



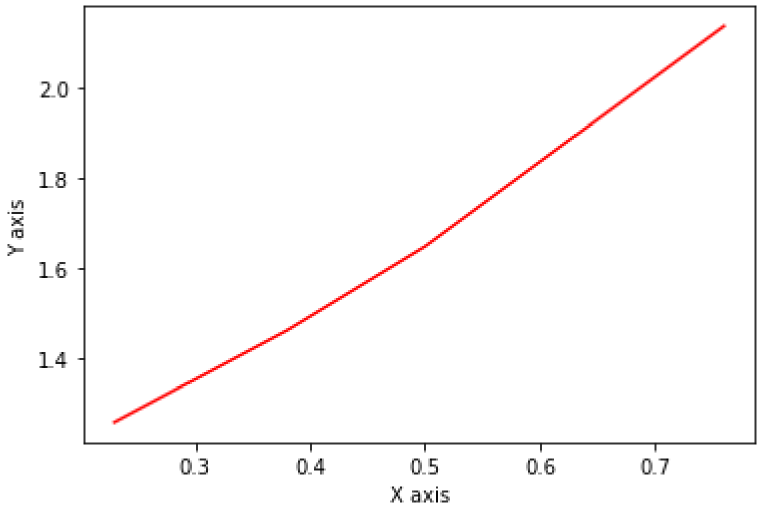
In [15]:
```
#expotential graph for dataset-1 using x
```

In [12]:
```
x = data1['in-vitro']
def exp_func(x):
    return np.exp(x)

# calculate y values using the exponential function
y = exp_func(x)

# plot the graph
plt.plot(x, y, color = "blue")

# set x and y labels
plt.xlabel('X axis')
plt.ylabel('Y axis')

# show the graph
plt.show()
```



In [17]:
```
#expotential graph for dataset-2 using x
```

In [13]:
```
x = data2['in-vitro']
def exp_func(x):
    return np.exp(x)

# calculate y values using the exponential function
y = exp_func(x)

# plot the graph
plt.plot(x, y, color = "red")

# set x and y labels
plt.xlabel('X axis')
plt.ylabel('Y axis')

# show the graph
plt.show()
```



In [19]:
```
#for Data-1 training and spliting
```

In [41]:
```
from sklearn.model_selection import train_test_split
X = data1[['in-vitro']].values
y = data1[['in-vivo']].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(16, 1) (5, 1) (16, 1) (5, 1)
```

In [21]:
```
#testing
```

In [42]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
y_pred = lin_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

```
Mean Squared Error: 0.009988961891974396
R-squared: 0.8716409532553884
```

In [43]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

# Convert the target variable to discrete class labels using LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)

# Transform the test labels using the fitted encoder
y_test = le.transform(y_test)

# Create a logistic regression object and fit it to the training data
log_reg = LogisticRegression()
log_reg.fit(X_train, y_train)

# Predict the class labels for the testing data
y_pred = log_reg.predict(X_test)

# Calculate the accuracy of the model
accuracy = log_reg.score(X_test, y_test)

# Print the accuracy
print('Accuracy:', accuracy)
```

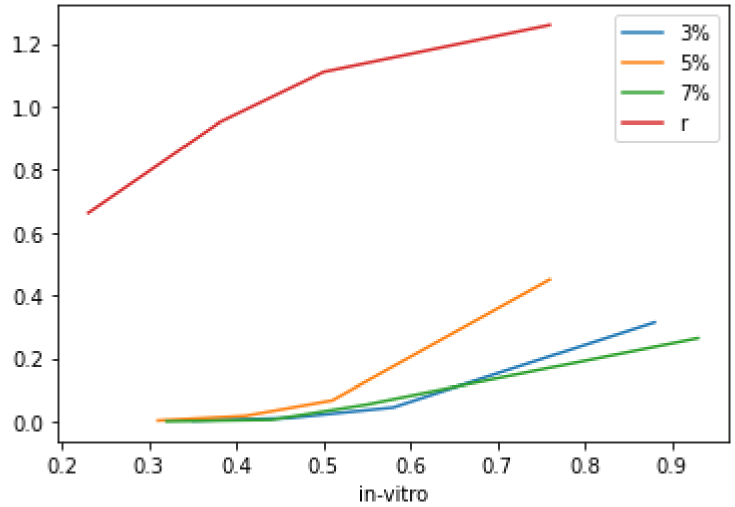
```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:115: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
C:\Users\HP\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:133: DataConversionWarning: A column-vector
y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[43], line 9
      6 y_train = le.fit_transform(y_train)
      8 # Transform the test labels using the fitted encoder
----> 9 y_test = le.transform(y_test)
     11 # Create a logistic regression object and fit it to the training data
     12 log_reg = LogisticRegression()

File ~\anaconda3\lib\site-packages\sklearn\preprocessing\_label.py:138, in LabelEncoder.transform(self, y)
    135 if _num_samples(y) == 0:
    136     return np.array([])
--> 138 return _encode(y, uniques=self.classes_)

File ~\anaconda3\lib\site-packages\sklearn\utils\_encode.py:189, in _encode(values, uniques, check_unknown)
    187     diff = _check_unknown(values, uniques)
    188     if diff:
--> 189         raise ValueError(f"y contains previously unseen labels: {str(diff)}")
    190 return np.searchsorted(uniques, values)

ValueError: y contains previously unseen labels: [0.002856, 0.011032, 0.718]
```

In [ ]:
```
#Slope on basis of type
```

In [45]:
```
import numpy as np
from sklearn.linear_model import LinearRegression

for type, group in groups:
    # Get the X and Y columns for the current group
    x_col = group["in-vitro"].values
    y_col = group["in-vivo"].values
    # Take the logarithm of both x and y
    log_x = np.log(x_col)
    log_y = np.log(y_col)
    if len(x_col) >= 2:
        slope, intercept = np.polyfit(x_col,y_col, 1)
        print(f"type {type}: slope = {slope} intercept = {intercept}")

fig, ax = plt.subplots()
for key, group in groups:
    group.plot(ax=ax, x='in-vitro', y='in-vivo', label=key)
ax.legend()
plt.show()
```

```
type 3%: slope = 0.6189083612573799 intercept = -0.2580067450135633
type 5%: slope = 1.0450377653631286 intercept = -0.3850892882681565
type 7%: slope = 0.4605791866028711 intercept = -0.176788844497607 93
type r: slope = 1.0877965322714451 intercept = 0.4866663711630993
```



In [ ]:
```
#using power function to pred value
```

In [49]:
```
#for type: 3%
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def power_func(x, a, b):
    return a * np.power(x, b)

# Generate some sample data
x = np.array([0.35,0.46,0.58,0.88])
y = np.array([0.002856,0.011032,0.044717,0.31429])

coefficients = np.polyfit(np.log(x), np.log(y), 1)
a = np.exp(coefficients[1])
b = coefficients[0]
# Calculate the R-squared value
y_pred = power_func(x, a, b)
print(a,b)
print(y_pred)
print("The equation of the power function that fits the data points is: y = {:.4f}x^({:.4f})".format(a, b))
```

```
0.6412496980205362 5.143352955497148
[0.0028974  0.01181603 0.03892726 0.33226298]
The equation of the power function that fits the data points is: y = 0.6412x^(5.1434)
```

In [47]:
```
y = [0.002856,0.011032,0.044717,0.31429]
y_pred=[0.0028974, 0.01181603,0.03892726, 0.33226298]
r2_sq = r2_score(y,y_pred)
print(r2_sq)
```

```
0.9946002647878214
```

In [ ]:
```
#predicting
```

In [17]:
```
#rechecking
x = np.array([0.35,0.46,0.58,0.88])
y = 0.6412*np.power(x,5.1434)
print(y)
```

```
[0.00289703 0.01181468 0.03892325 0.33223524]
```

In [55]:
```
#lower bound
import pandas as pd
data3= pd.read_csv('D:/class/Project/ML/sensitivity analysis/updatedataset.csv')
```

In [ ]:
```
#for type = 5%
```

In [51]:
```
#for type: 5%
import numpy as np
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def power_func(x, a, b):
    return a * np.power(x, b)

# Generate some sample data
x = np.array([0.31,0.41,0.51,0.76])
y = np.array([0.003526,0.017983,0.067054,0.450705])

coefficients = np.polyfit(np.log(x), np.log(y), 1)
a = np.exp(coefficients[1])
b = coefficients[0]
# Calculate the R-squared value
y_pred = power_func(x, a, b)
print(a,b)
print(y_pred)
print("The equation of the power function that fits the data points is: y = {:.4f}x^({:.4f})".format(a, b))
```

```
2.1956582285632407 5.417939800070449
[0.00385295 0.01752472 0.05717387 0.4963865 ]
The equation of the power function that fits the data points is: y = 2.1957x^(5.4179)
```

In [30]:
```
#rechecking
x = np.array([0.31,0.41,0.51,0.76])
y = 2.1957*np.power(x,5.4179)
print(y)
```

```
[0.0038532  0.01752567 0.05717649 0.49640136]
```

In [ ]:
```
#for 7%
```

In [53]:
```
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def power_func(x, a, b):
    return a * np.power(x, b)

# Generate some sample data
x = np.array([0.32,0.44,0.55,0.93])
y = np.array([0.0008,0.005723,0.054019,0.264])

coefficients = np.polyfit(np.log(x), np.log(y), 1)
a = np.exp(coefficients[1])
b = coefficients[0]
# Calculate the R-squared value
y_pred = power_func(x, a, b)
print(a,b)
print(y_pred)
print("The equation of the power function that fits the data points is: y = {:.4f}x^({:.4f})".format(a, b))
```

```
0.59424662503013 5.497346558731055
[0.00113137 0.00651482 0.02221521 0.39875539]
The equation of the power function that fits the data points is: y = 0.5942x^(5.4973)
```

In [36]:

```
#rechecking
x = np.array([0.32,0.44,0.55,0.93])
y = 0.5942*np.power(x,5.4973)
print(y)
```

```
[0.00113134 0.00651456 0.02221409 0.39872545]
```

In [11]:

```
#for updates dataset
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
import pandas as pd

data_ref = pd.read_csv('D:/class/Project/ML/sensitivity analysis/updatedataset.csv')
def power_func(x, a, b):
    return a * np.power(x, b)

x = data_ref['in-vitro']
y = data_ref['in-vivo']
coefficients = np.polyfit(np.log(x), np.log(y), 1)
a = np.exp(coefficients[1])
b = coefficients[0]
# Calculate the R-squared value
y_pred = power_func(x, a, b)
print(a,b)
print(y_pred)
print("The equation of the power function that fits the data points is: y = {:.4f}x^({:.4f})".format(a, b))
```

```
0.4310313419771757 2.6854056658286236
0     0.025713
1     0.053564
2     0.099820
3     0.305789
4     0.018561
5     0.039326
6     0.070667
7     0.206274
8     0.020213
9     0.047537
10    0.086552
11    0.354709
12    0.008327
13    0.032067
14    0.067007
15    0.206274
Name: in-vitro, dtype: float64
The equation of the power function that fits the data points is: y = 0.4310x^(2.6854)
```

In [57]:
```
x = np.array([0.2,0.32,0.37,0.6,0.35,0.55,0.62,0.8,0.31,0.42,0.53,0.79])
y = 0.431*np.power(x,2.6854)
print(y)
```

```
[0.00572089 0.02021189 0.02984875 0.1093263  0.02571092 0.08654605
 0.11938944 0.23672009 0.01856009 0.04195183 0.07835157 0.22885745]
```

In [ ]:
```
#Ref dataset predict
```

In [6]:
```
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score

def power_func(x, a, b):
    return a * np.power(x, b)

x = np.array([0.23,0.38,0.5,0.76])
y = np.array([0.662684,0.949331,1.109712,1.259118])
coefficients = np.polyfit(np.log(x), np.log(y), 1)
a = np.exp(coefficients[1])
b = coefficients[0]
# Calculate the R-squared value
y_pred = power_func(x, a, b)
print(a,b)
print(y_pred)
print("The equation of the power function that fits the data points is: y = {:.4f}x^({:.4f})".format(a, b))
```

```
1.5382855938476874 0.5438184388786232
[0.69172393 0.90889883 1.05519156 1.32501626]
The equation of the power function that fits the data points is: y = 1.5383x^(0.5438)
```

In [7]:
```
x = np.array([0.23,0.38,0.5,0.76])
y = 1.5383*np.power(x,0.5438)
print(y)
```

```
[0.69174915 0.90892356 1.05521493 1.32503537]
```

In [ ]:

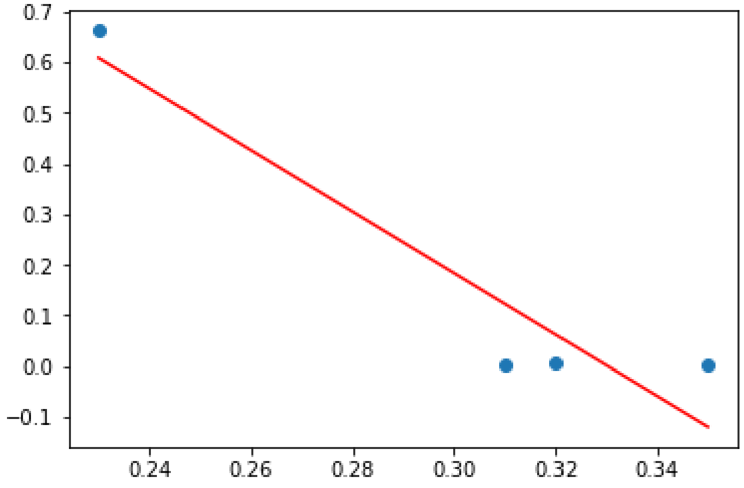
```
```

In [23]:

```
# making linear regressor graph and find slope using that regressor line
```

In [23]:
```
#for time = 2
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df_time2= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-2.csv')

# Create some sample data
x = df_time2['in-vitro'].values
y = df_time2['in-vivo'].values

# Fit the linear regression model
model = LinearRegression().fit(x.reshape(-1, 1), y)

# Print the intercept and slope of the regression line
print(f"Intercept: {model.intercept_:.2f}")
print(f"Slope: {model.coef_[0]:.2f}")

# Plot the data points and the regression line
plt.scatter(x, y)
plt.plot(x, model.predict(x.reshape(-1, 1)), color='red')
plt.show()
```

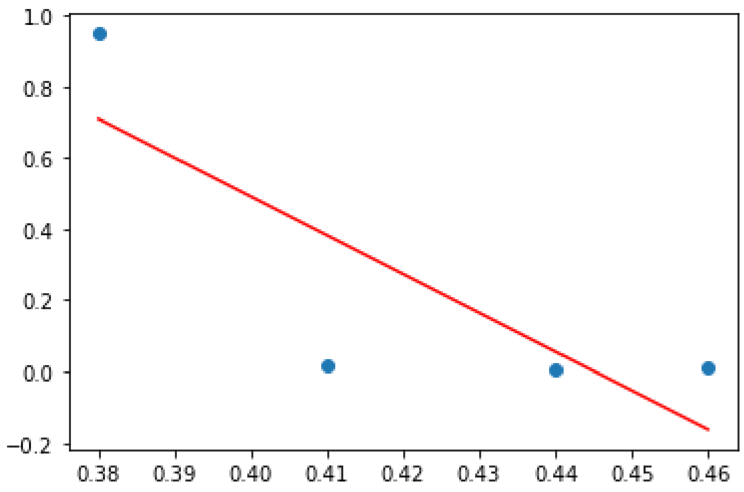
```
Intercept: 2.00
Slope: -6.07
```



In [25]:
```
#slope of time = 2.5
```

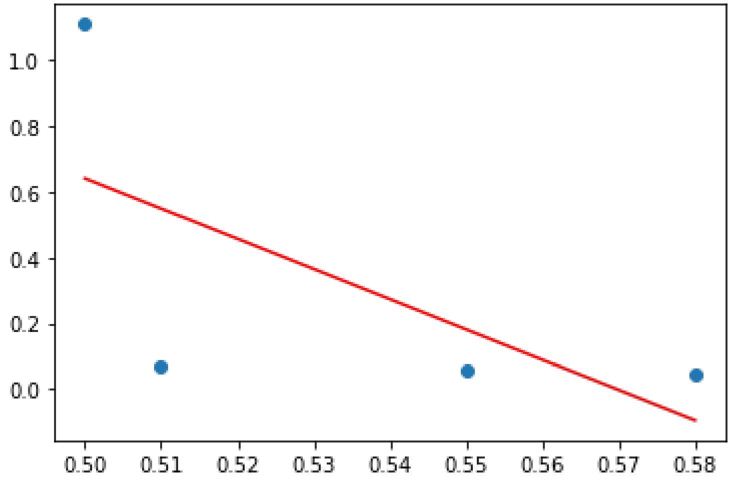
In [26]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df_time25= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-2.5.csv')

# Create some sample data
x = df_time25['in-vitro'].values
y = df_time25['in-vivo'].values

# Fit the linear regression model
model = LinearRegression().fit(x.reshape(-1, 1), y)

# Print the intercept and slope of the regression line
print(f"Intercept: {model.intercept_:.2f}")
print(f"Slope: {model.coef_[0]:.2f}")

# Plot the data points and the regression line
plt.scatter(x, y)
plt.plot(x, model.predict(x.reshape(-1, 1)), color='red')
plt.show()
```

```
Intercept: 4.85
Slope: -10.90
```



In [27]:
```python
#for time = 3

import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df_time3= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-3.csv')

# Create some sample data
x = df_time3['in-vitro'].values
y = df_time3['in-vivo'].values

# Fit the linear regression model
model = LinearRegression().fit(x.reshape(-1, 1), y)

# Print the intercept and slope of the regression line
print(f"Intercept: {model.intercept_:.2f}")
print(f"Slope: {model.coef_[0]:.2f}")

# Plot the data points and the regression line
plt.scatter(x, y)
plt.plot(x, model.predict(x.reshape(-1, 1)), color='red')
plt.show()
```

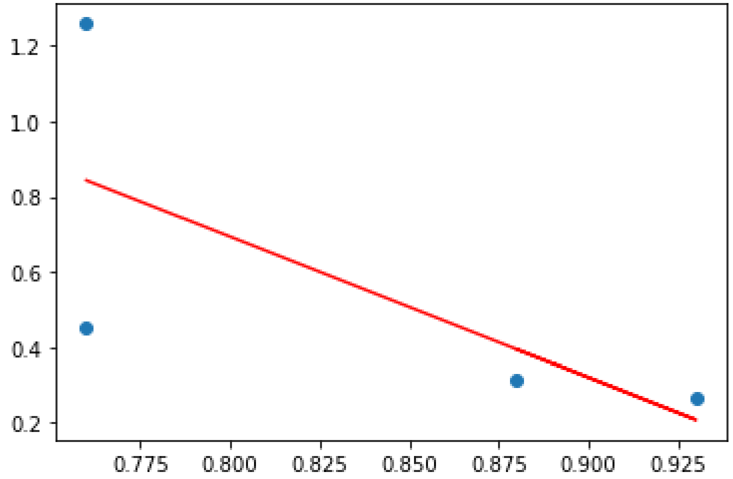
```
Intercept: 5.24
Slope: -9.19
```



In [28]:

```
#for time = 4
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

df_time4= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-4.csv')

# Create some sample data
x = df_time4['in-vitro'].values
y = df_time4['in-vivo'].values

# Fit the linear regression model
model = LinearRegression().fit(x.reshape(-1, 1), y)

# Print the intercept and slope of the regression line
print(f"Intercept: {model.intercept_:.2f}")
print(f"Slope: {model.coef_[0]:.2f}")

# Plot the data points and the regression line
plt.scatter(x, y)
plt.plot(x, model.predict(x.reshape(-1, 1)), color='red')
plt.show()
```

```
Intercept: 3.69
Slope: -3.74
```



In [29]:
```
#now predicting the values for above slopes
```

In [30]:
```
#for time = 2
slope = -6.07
intercept = 2.00
x = df_time2['in-vitro']
y = [(slope * xi + intercept) for xi in x]
# Print predicted y values
print(y)
```

```
[-0.12449999999999983, 0.057599999999999874, 0.11829999999999985, 0.6038999999999999]
```

In [31]:
```
#for time = 2.5
intercept= 4.85
slope = -10.90
x = df_time25['in-vitro']
y = [(slope * xi + intercept) for xi in x]
# Print predicted y values
print(y)
print(x)
```

```
[-0.1640000000000006, 0.05399999999999938, 0.38099999999999934, 0.7079999999999993]
0    0.46
1    0.44
2    0.41
3    0.38
Name: in-vitro, dtype: float64
```

In [32]:
```
#for time = 3
intercept= 5.24
slope= -9.19
x = df_time3['in-vitro']
y = [(slope * xi + intercept) for xi in x]
# Print predicted y values
print(y)
print(x)
```

```
[-0.09019999999999939, 0.18550000000000022, 0.5531000000000006, 0.6450000000000005]
0    0.58
1    0.55
2    0.51
3    0.50
Name: in-vitro, dtype: float64
```

In [33]:
```
#for time = 4
intercept = 3.69
slope= -3.74
x = df_time4['in-vitro']
y = [(slope * xi + intercept) for xi in x]
# Print predicted y values
print(y)
print(x)
```

```
[0.3987999999999996, 0.21179999999999977, 0.8475999999999999, 0.8475999999999999]
0    0.88
1    0.93
2    0.76
3    0.76
Name: in-vitro, dtype: float64
```

In [34]:
```
# mean intercept, in-vitro and in-vivo value for slope
#calculated slope should lie b\w -3 to -10
#mean median for intecept
```

In [3]:
```
#mean of intercept
intercept = (2+4.85+5.24+3.69)/4
print(intercept)

x = np.array([0.35, 0.32, 0.31, 0.23, 0.46,0.44,0.41,0.38, 0.58, 0.55, 0.51, 0.50,0.88,0.93,0.76,0.76])
y = np.array([0.001, 0.057599999999999874, 0.11829999999999985, 0.6038999999999999, 0.001640000000000006, 0.0539999999

slope = (y- intercept) / x

print("slope for each pair x,y values:")
print(slope)

mean_slope = np.mean(slope)

print("mean slope:", mean_slope)
```

```
3.945
slope for each pair x,y values:
[-11.26857143 -12.148125   -12.34419355 -14.52652174  -8.57252174
  -8.84318182  -8.69268293  -8.51842105  -6.79844483  -6.83545455
  -6.65078431  -6.6         -4.02977273  -4.01419355  -4.07552632
  -4.07552632]
mean slope: -7.999620115429193
```

In [36]:
```
#predict values
```

In [ ]:
```
#predicting test values on basis of time =2,2.5,3,4
```

In [38]:
```
#for time = 2
slope = -6.07
intercept = 2.00
x = np.array([0.2,0.35,0.31])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 0.786  -0.1245  0.1183]
```

In [39]:
```
#time = 2.5
intercept= 4.85
slope = -10.90
x = np.array([0.32,0.55,0.42])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 1.362 -1.145  0.272]
```

In [40]:
```
#for time = 3
intercept= 5.24
slope= -9.19
x = np.array([0.37,0.62,0.53])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 1.8397 -0.4578  0.3693]
```

In [43]:
```
#for time = 4
intercept = 3.69
slope= -3.74
x = np.array([0.6,0.8,0.79])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[1.446  0.698  0.7354]
```

In [ ]:
```
#predicting on basis of type = test,lower, upper
```

In [42]:
```
#for every x value
#predict test
intercept= 3.945
slope = -8.054461048980
x = np.array([0.31,0.42,0.53,0.79])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 1.44811707  0.56212636 -0.32386436 -2.41802423]
```

In [88]:
```
#for lower bound
intercept= 3.945
slope = -8.054461048980
x = np.array([0.2,0.32,0.37,0.6])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 2.33410779  1.36757246  0.96484941 -0.88767663]
```

In [94]:
```
#for upper bound
intercept= 3.945
slope = -8.054461048980
x = np.array([0.2,0.32,0.37,0.6])
y = (slope * x + intercept)
# Print predicted y values
print(y)
```

```
[ 2.33410779  1.36757246  0.96484941 -0.88767663]
```

In [28]:
```
data3 = pd.read_csv('D:/class/Project/ML/sensitivity analysis/updatedataset.csv')
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
#Bootstraping
```

In [27]:
```
!pip install mlbootstrap <= 0.0.4
```

```
Requirement already satisfied: mlbootstrap in c:\users\hp\anaconda3\lib\site-packages (0.0.5)
Requirement already satisfied: PyYAML>=3.12 in c:\users\hp\anaconda3\lib\site-packages (from mlbootstrap) (6.0)
Requirement already satisfied: fire==0.1.1 in c:\users\hp\anaconda3\lib\site-packages (from mlbootstrap) (0.1.1)
Requirement already satisfied: six in c:\users\hp\anaconda3\lib\site-packages (from fire==0.1.1->mlbootstrap) (1.16.
0)
Requirement already satisfied: ipython<6.0 in c:\users\hp\anaconda3\lib\site-packages (from fire==0.1.1->mlbootstra
p) (5.10.0)
Requirement already satisfied: pygments<2.6 in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==0.1.
1->mlbootstrap) (2.5.2)
Requirement already satisfied: simplegeneric>0.8 in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire=
=0.1.1->mlbootstrap) (0.8.1)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==0.1.1->m
lbootstrap) (0.4.4)
Requirement already satisfied: traitlets>=4.2 in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==0.
1.1->mlbootstrap) (5.1.1)
Requirement already satisfied: decorator in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==0.1.1->
mlbootstrap) (5.1.1)
Requirement already satisfied: pickleshare in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==0.1.1
->mlbootstrap) (0.7.5)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in c:\users\hp\anaconda3\lib\site-packages (from ipython
<6.0->fire==0.1.1->mlbootstrap) (1.0.18)
Requirement already satisfied: setuptools>=18.5 in c:\users\hp\anaconda3\lib\site-packages (from ipython<6.0->fire==
0.1.1->mlbootstrap) (61.2.0)
Requirement already satisfied: wcwidth in c:\users\hp\anaconda3\lib\site-packages (from prompt-toolkit<2.0.0,>=1.0.4
->ipython<6.0->fire==0.1.1->mlbootstrap) (0.2.5)
```

In [17]:
```
!pip install bootstrap
```

```
Requirement already satisfied: bootstraps in c:\users\hp\anaconda3\lib\site-packages (1.0.1)
```

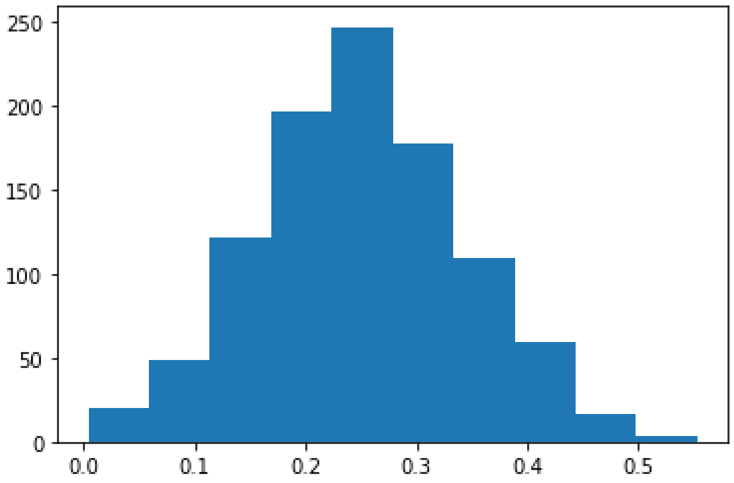
In [2]:
```
!pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\hp\anaconda3\lib\site-packages (23.0.1)
```

In [19]:
```
import bootstraps
from mlbootstrap import bootstraps
import pandas as pd
df_time2= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-2.csv')
# Define a resampling function that returns a new dataset with the same x,y coordinates
# but with new function values generated by adding random noise to the original function values
def resample(df_time2):
    df_time2_new = df_time2.copy()
    df_time2_new['f'] += np.random.normal(loc=0, scale=0.1, size=len(df_time2_new))
    return df_new

# Generate 100 bootstrap samples
boot_samples = mlbootstrap.bootstraps(df_time2, num_boots=100)

# Resample each bootstrap sample to obtain a new dataset
boot_datasets = [resample(sample) for sample in boot_samples]

# Concatenate all the new datasets to obtain the final bootstrap dataset
boot_df = pd.concat(boot_datasets)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[19], line 2
      1 import bootstraps
----> 2 from mlbootstrap import bootstraps
      3 import pandas as pd
      4 df_time2= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-2.csv')

ImportError: cannot import name 'bootstraps' from 'mlbootstrap' (C:\Users\HP\anaconda3\lib\site-packages\mlbootstrap
\__init__.py)
```

In [77]:
```
#bootstrap for t-2
import random
import statistics
df_time2= pd.read_csv('D:/class/Project/ML/sensitivity analysis/time-2.csv')
df_time2 = df_time2.drop(0)
df_time2 = df_time2.values.ravel()
def myboot(mysample):
    resample = random.choices(mysample, k = len(mysample))
    m = statistics.mean(resample)
    return m

mymean = []
for i in range(1000):
    x= myboot(df_time2)
    mymean.append(x)

plt.hist(mymean)
plt.show()
```



In [79]:
```
mymean.sort()
print("lower bound 95 confidencelevel cutoff",mymean[24])
print("upper bound 95 confidencelevel cutoff",mymean[974])
print("middle 95 confidencelevel cutoff",mymean[499])
```

```
lower bound 95 confidencelevel cutoff 0.07984200000000001
upper bound 95 confidencelevel cutoff 0.437342
middle 95 confidencelevel cutoff 0.25536833333333336
```

In [ ]:
```
#motecarlo for dataset
```