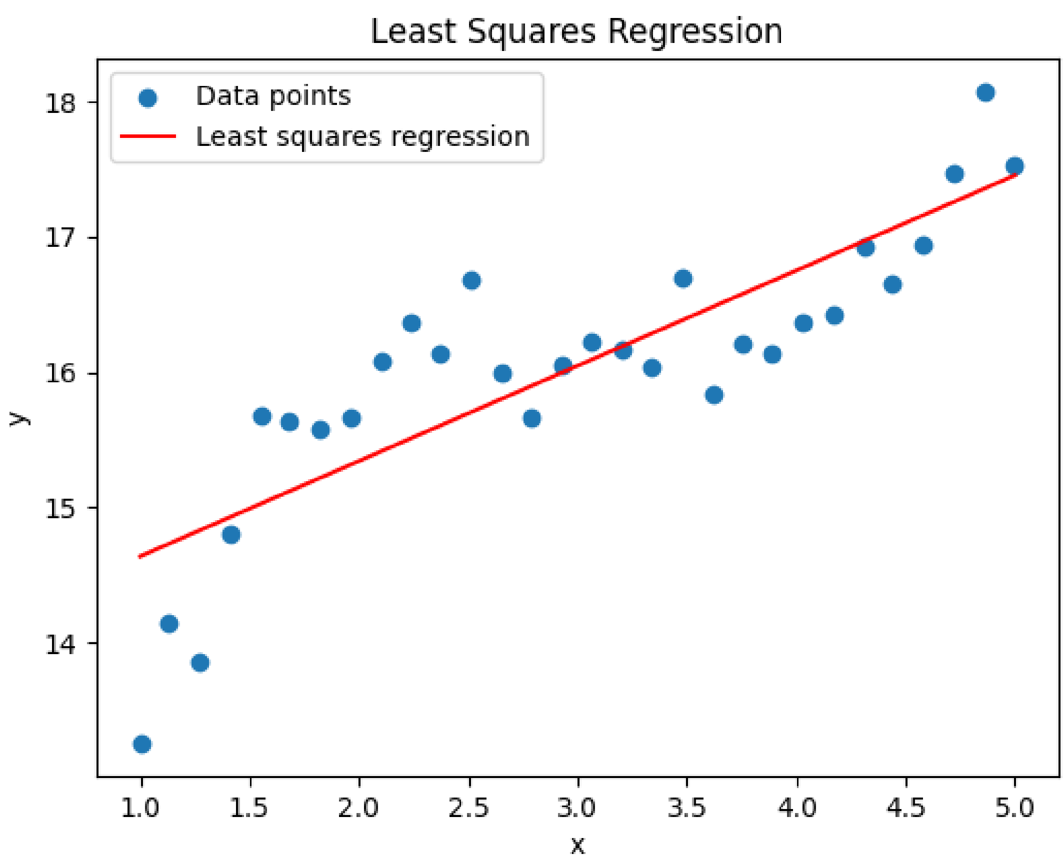
```
#Problem 3
#1.

import numpy as np

# Given lists x and y
x = [1.00, 1.13, 1.27, 1.41, 1.55, 1.68, 1.82, 1.96, 2.10, 2.24, 2.37,
2.51, 2.65,
     2.79, 2.93, 3.06, 3.20, 3.34, 3.48, 3.62, 3.75, 3.89, 4.03, 4.17,
4.31, 4.44,
     4.58, 4.72, 4.86, 5.00]

y = [13.26, 14.15, 13.86, 14.81, 15.68, 15.64, 15.58, 15.67, 16.08,
16.36, 16.14, 16.68, 16.00, 15.66, 16.05, 16.22, 16.17, 16.03, 16.69,
15.83, 16.21, 16.13, 16.36, 16.42, 16.92, 16.65, 16.94, 17.47, 18.07,
17.52]

# Fit a line to the data
coefficients = np.polyfit(x, y, 1)

# The coefficients are returned in the order of highest degree to
lowest,
# so the slope of the line is the first element and the y-intercept is
the second.
slope, intercept = coefficients

print(f"The equation of the least squares line is: y = {slope} * x +
{intercept}")
```

```
The equation of the least squares line is: y = 0.7029398624005565 * x
+ 13.936127465489536
```

```
#Problem 3
#2.
import numpy as np
import matplotlib.pyplot as plt


# Given lists x and y
# Convert lists x and y to numpy arrays
x = np.array([1.00, 1.13, 1.27, 1.41, 1.55, 1.68, 1.82, 1.96, 2.10,
2.24, 2.37, 2.51, 2.65,
     2.79, 2.93, 3.06, 3.20, 3.34, 3.48, 3.62, 3.75, 3.89, 4.03, 4.17,
4.31, 4.44,
     4.58, 4.72, 4.86, 5.00])

y = np.array([13.26, 14.15, 13.86, 14.81, 15.68, 15.64, 15.58, 15.67,
16.08, 16.36, 16.14, 16.68, 16.00, 15.66, 16.05, 16.22, 16.17, 16.03,
16.69, 15.83, 16.21, 16.13, 16.36, 16.42, 16.92, 16.65, 16.94, 17.47,
18.07, 17.52])
```

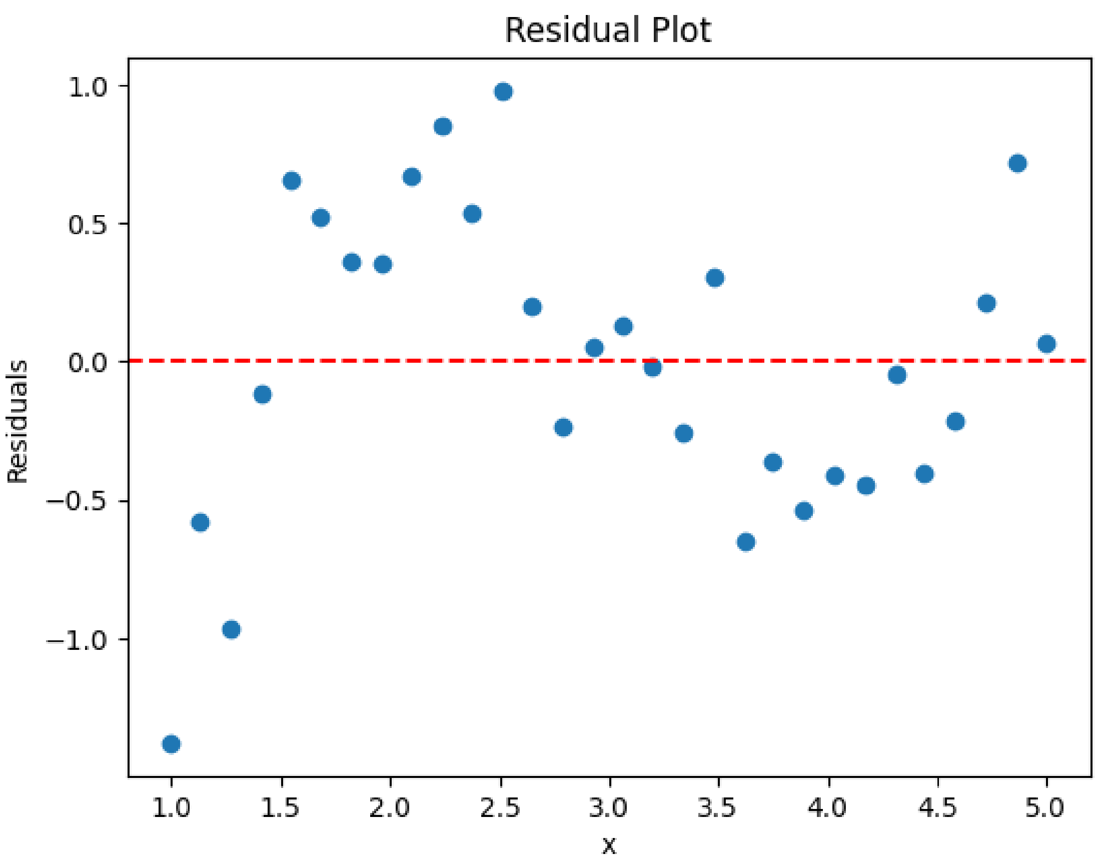
```python
# Assemble matrix A
A = np.vstack([x, np.ones(len(x))]).T

# Perform least squares regression
alpha = np.dot(np.dot(np.linalg.inv(np.dot(A.T, A)), A.T), y)
print("Coefficients (α1, α2):", alpha)

# Plot data points and regression line
plt.scatter(x, y, label="Data points")
plt.plot(x, alpha[0] * x + alpha[1], color="red", label="Least squares regression")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Least Squares Regression")
plt.legend()
plt.show()
```

```
Coefficients (α1, α2): [ 0.70293986 13.93612747]
```




```python
#Problem 3
#4.
```

```
import matplotlib.pyplot as plt

# Calculate the predicted values
y_pred = alpha[0] * x + alpha[1]

# Calculate the residuals
residuals = y - y_pred

# Create the residual plot
plt.scatter(x, residuals)
plt.axhline(0, color='red', linestyle='--')  # Add a horizontal line
at y=0
plt.xlabel('x')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()

#model 1. is suitable
```



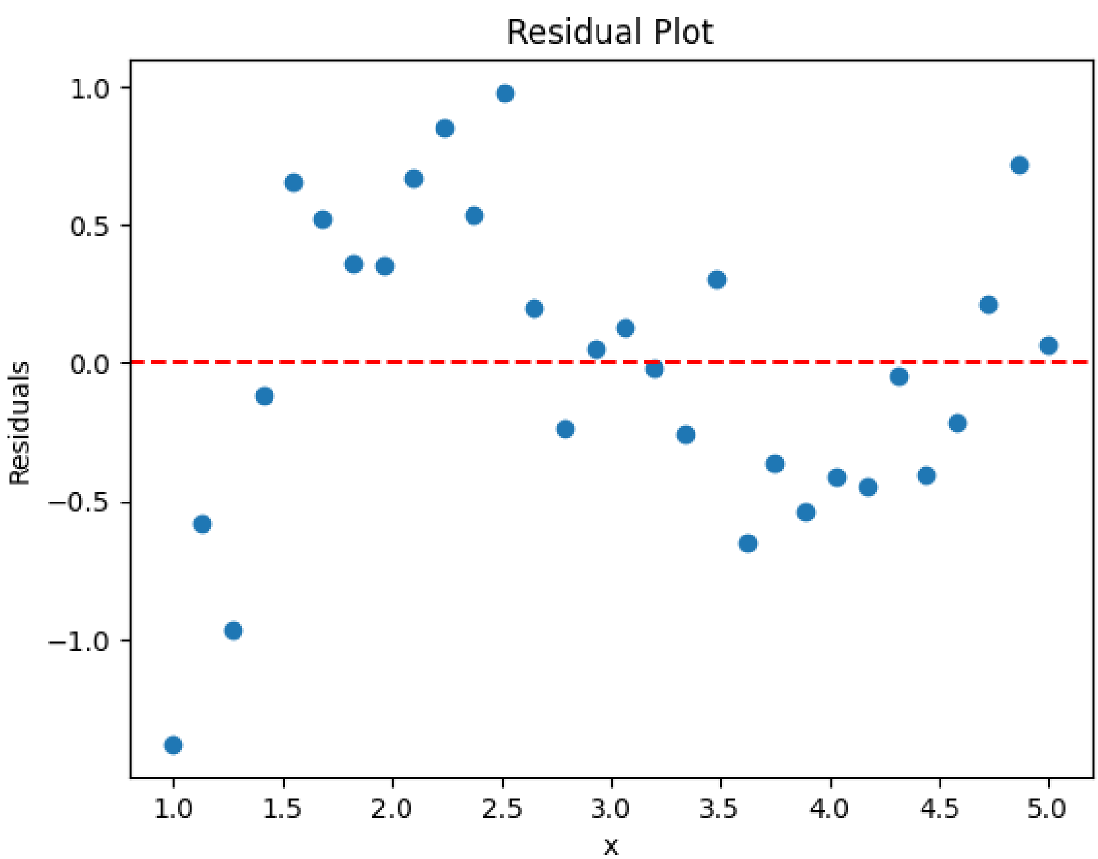

```
#Problem 3
#3.

import numpy as np
```

```
import scipy.stats as stats

# Given lists x and y
x = np.array([1.00, 1.13, 1.27, 1.41, 1.55, 1.68, 1.82, 1.96, 2.10,
2.24, 2.37, 2.51, 2.65,
     2.79, 2.93, 3.06, 3.20, 3.34, 3.48, 3.62, 3.75, 3.89, 4.03, 4.17,
4.31, 4.44,
     4.58, 4.72, 4.86, 5.00])

y = np.array([13.26, 14.15, 13.86, 14.81, 15.68, 15.64, 15.58, 15.67,
16.08, 16.36, 16.14, 16.68, 16.00, 15.66, 16.05, 16.22, 16.17, 16.03,
16.69, 15.83, 16.21, 16.13, 16.36, 16.42, 16.92, 16.65, 16.94, 17.47,
18.07, 17.52])

# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Calculate the t critical value for 95% confidence level
t_critical = stats.t.ppf((1 + 0.95) / 2., len(x) - 2)

# Calculate the confidence intervals for the slope
confidence_interval = [slope - t_critical * std_err, slope +
t_critical * std_err]

print(f"The 95% confidence intervals for the slope (β) are:
{confidence_interval}")
```

```
The 95% confidence intervals for the slope (β) are:
[0.5269975559927793, 0.8788821688083326]
```

```
#Problem 3
#5.

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
import scipy.stats as stats

# Given lists x and y
x = np.array([1.00, 1.13, 1.27, 1.41, 1.55, 1.68, 1.82, 1.96, 2.10,
2.24, 2.37, 2.51, 2.65,
     2.79, 2.93, 3.06, 3.20, 3.34, 3.48, 3.62, 3.75, 3.89, 4.03, 4.17,
4.31, 4.44,
     4.58, 4.72, 4.86, 5.00])

y = np.array([13.26, 14.15, 13.86, 14.81, 15.68, 15.64, 15.58, 15.67,
16.08, 16.36, 16.14, 16.68, 16.00, 15.66, 16.05, 16.22, 16.17, 16.03,
16.69, 15.83, 16.21, 16.13, 16.36, 16.42, 16.92, 16.65, 16.94, 17.47,
18.07, 17.52])
```

```
# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Calculate the predicted values
y_pred = slope * x + intercept

# Calculate the residuals
residuals = y - y_pred

# Create the residual plot
plt.scatter(x, residuals)
plt.axhline(0, color='red', linestyle='--')  # Add a horizontal line
at y=0
plt.xlabel('x')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()

# Calculate and print R^2
r2 = r2_score(y, y_pred)
print(f'R^2: {r2}')
```




```
R^2: 0.7051930538528302
```

```python
#Problem 3
#6.
import numpy as np
import scipy.stats as stats

# Given lists x and y
x = np.array([1.00, 1.13, 1.27, 1.41, 1.55, 1.68, 1.82, 1.96, 2.10,
2.24, 2.37, 2.51, 2.65,
     2.79, 2.93, 3.06, 3.20, 3.34, 3.48, 3.62, 3.75, 3.89, 4.03, 4.17,
4.31, 4.44,
     4.58, 4.72, 4.86, 5.00])

y = np.array([13.26, 14.15, 13.86, 14.81, 15.68, 15.64, 15.58, 15.67,
16.08, 16.36, 16.14, 16.68, 16.00, 15.66, 16.05, 16.22, 16.17, 16.03,
16.69, 15.83, 16.21, 16.13, 16.36, 16.42, 16.92, 16.65, 16.94, 17.47,
18.07, 17.52])

# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Calculate the predicted values
y_pred = slope * x + intercept

# Calculate the residuals
residuals = y - y_pred

# Calculate the mean squared error (MSE)
MSE = np.mean(residuals**2)

# Calculate the mean of x
x_mean = np.mean(x)

# Calculate the sum of squares of x
x_SS = np.sum((x - x_mean)**2)

# Points at which we want to estimate the response
x_star = np.array([1.2, 4.6])

# Calculate the predicted responses at x_star
y_star = slope * x_star + intercept

# Calculate the standard errors of the mean response at x_star
SE_star = np.sqrt(MSE * (1/len(x) + (x_star - x_mean)**2 / x_SS))

# Calculate the t critical value for 95% confidence level
t_critical = stats.t.ppf((1 + 0.95) / 2., len(x) - 2)

# Calculate the confidence intervals for the mean response at x_star
CI_star = y_star - t_critical * SE_star, y_star + t_critical * SE_star
```

```python
print(f"The 95% confidence intervals for the mean response at x* =
{x_star} are: {CI_star}")
```

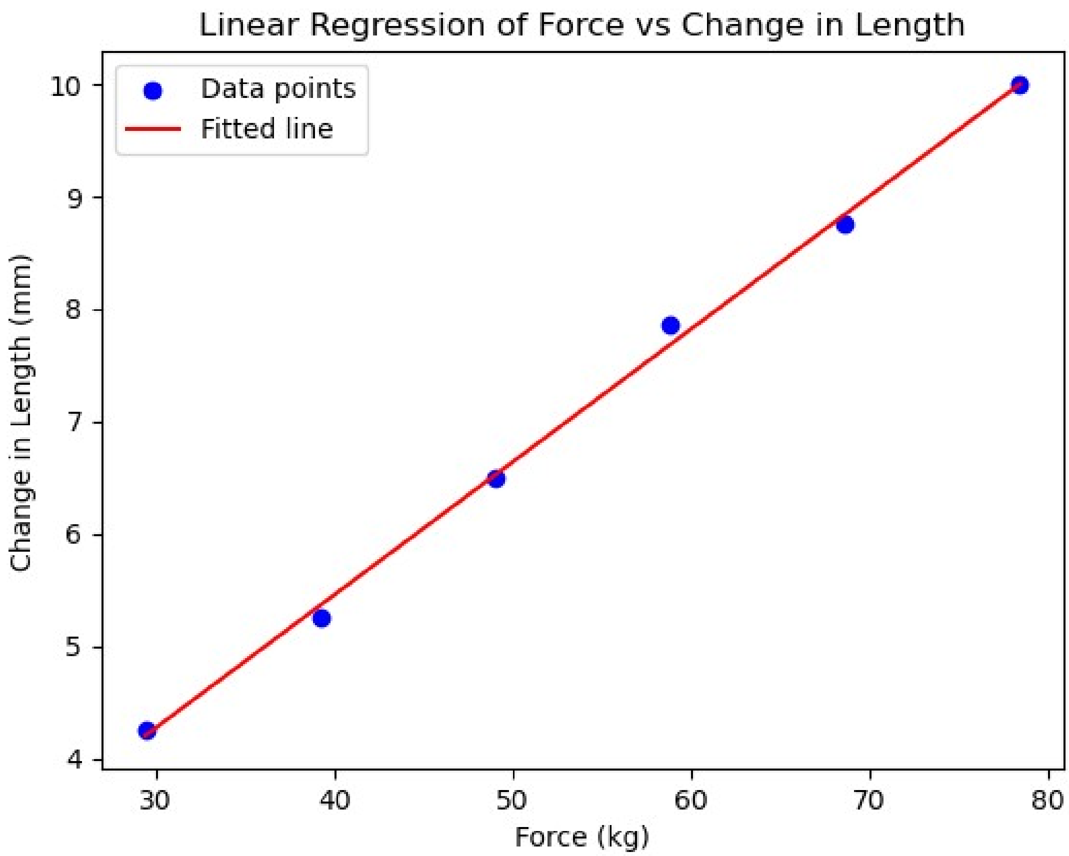
```
The 95% confidence intervals for the mean response at x* = [1.2 4.6]
are: (array([14.41313186, 16.82963653]), array([15.14617874,
17.50966513]))
```

```python
#HW8 problem 1


import numpy as np


neutral = [0, 2, 0, 1, 0.5, 0, 0.5, 2, 1, 0, 0, 0, 0, 1]
sad = [3, 4, 0.5, 1, 2.5, 2, 1.5, 0, 1, 1.5, 1.5, 2.5, 4, 3, 3.5, 1,
3.5]

mean_nt = np.mean(neutral)
std_nt = np.std(neutral)
print(len(neutral))
print("Neutral mean: ", mean_nt)
print("Neutral std: ", std_nt)
mean_sad = np.mean(sad)
std_sad = np.std(sad)
print(len(sad))
print("Sad mean: ", mean_sad)
print("Sad std: ", std_sad)
```

```
14
Neutral mean:  0.5714285714285714
Neutral std:  0.7034898429854362
17
Sad mean:  2.1176470588235294
Sad std:  1.2069579134519526
```

```python
#Problem 4
#1.

import numpy as np
from scipy import stats
import matplotlib.pyplot as plt


# Define the data
x = np.array([29.4, 39.2, 49.0, 58.8, 68.6, 78.4])  # Force in kg
y = np.array([4.25, 5.25, 6.5, 7.85, 8.75, 10.00])  # Change in length
in mm

# Fit the model using numpy.polyfit
beta1, beta0 = np.polyfit(x, y, 1)
```

```
print(f"β0 = {beta0}, β1 = {beta1}")


# Create a scatter plot of the data
plt.scatter(x, y, color='blue', label='Data points')
# Create a line plot of the fitted model
plt.plot(x, beta0 + beta1 * x, color='red', label='Fitted line')

# Add labels and title
plt.xlabel('Force (kg)')
plt.ylabel('Change in Length (mm)')
plt.title('Linear Regression of Force vs Change in Length')
plt.legend()

# Show the plot
plt.show()


#The line equation is Y = 0.7200000000000034 + 0.11836734693877549x

β0 = 0.7200000000000034, β1 = 0.11836734693877549
```



```python
#Problem 4
#2.

import numpy as np

from scipy import stats


# Define the data
x = np.array([29.4, 39.2, 49.0, 58.8, 68.6, 78.4])  # Force in kg
y = np.array([4.25, 5.25, 6.5, 7.85, 8.75, 10.00])  # Change in length
in mm

# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Calculate the t-value for a 95% confidence interval (df = n-2)
t = stats.t.ppf(1 - 0.05 / 2, df=len(x) - 2)

# Calculate the confidence interval
ci_low = slope - t * std_err
ci_high = slope + t * std_err

print(f"The 95% confidence interval for the slope is [{ci_low},
{ci_high}]")
```

```
The 95% confidence interval for the slope is [0.11064559817367174,
0.12608909570387924]
```

```python
#Problem 4
#3. 4. 5.
import numpy as np
from scipy import statsWhat

# Define the data
x = np.array([29.4, 39.2, 49.0, 58.8, 68.6, 78.4])  # Force in kg
y = np.array([4.25, 5.25, 6.5, 7.85, 8.75, 10.00])  # Change in length
in mm

# Perform linear regression
slope, intercept, r_value, p_value, std_err = stats.linregress(x, y)

# Calculate the t-statistic for the intercept
t_statistic = intercept / std_err

# Calculate the p-value for the t-statistic
p_value = 2 * (1 - stats.t.cdf(abs(t_statistic), df=len(x) - 2))

print(f"The t-statistic is {t_statistic} and the p-value is
{p_value}")
```

```
#compare the P value with significance level (0.05)

print("Is p_value > significance level (0.05): ", p_value > 0.05)


# since p_value is smaller than the significance level, we reject the
# null hypothesis which is the slope the the regression line is not
equal to 0
```

```
The t-statistic is 258.8844232768247 and the p-value is
1.3356244998874445e-09
Is p_value > significance level (0.05):  False
```

```
#Problem 5
# Group 1: Forest Area and IBI[^1^][1]
forest_area = [0, 0, 0, 0, 0, 0, 3, 3, 7, 8, 9, 10, 10, 11, 14, 17,
17, 18, 21, 22,
25, 31, 32, 33, 33, 33, 39, 41, 43, 43, 47, 49, 49, 52, 52, 59, 63,
68, 75, 79, 79, 80, 86, 89, 90, 95, 95, 100, 100]


ibi = [47, 61, 39, 59, 72, 76, 85, 89, 74, 89, 33, 46, 32, 80, 80, 78,
53, 43, 88, 84,
62, 55, 29, 29, 54, 78, 71, 55, 58, 71, 33, 59, 81, 71, 75, 64, 41,
82, 60, 84, 83, 82, 82, 86, 79, 67, 56, 85, 91]

# Group 2: Watershed Area
watershed_area = [21, 29, 31, 32, 34, 34, 49, 52, 2, 70, 6, 28, 21,
59, 69, 47, 8, 8, 58, 54, 10, 57,
18, 19, 39, 49, 9, 5, 14, 9, 23, 31, 18, 16, 21, 32, 10, 26, 9, 54,
12, 21, 27, 23, 26, 16, 26, 26, 28]

print(len(forest_area), len(ibi), len(watershed_area))
```

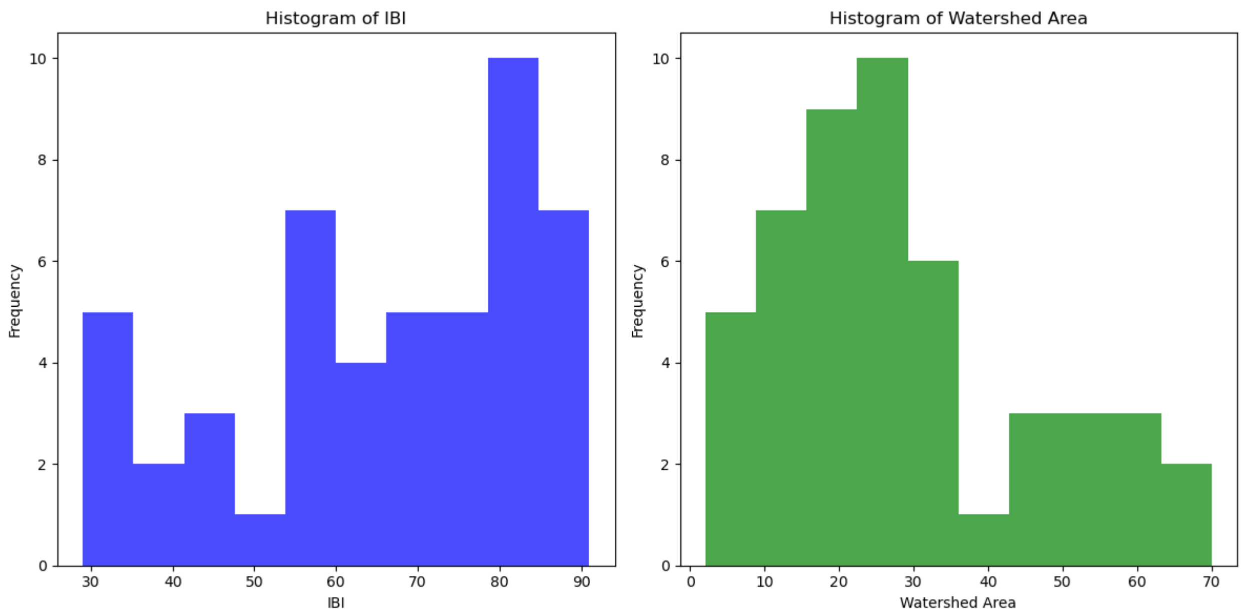
```
49 49 49
```

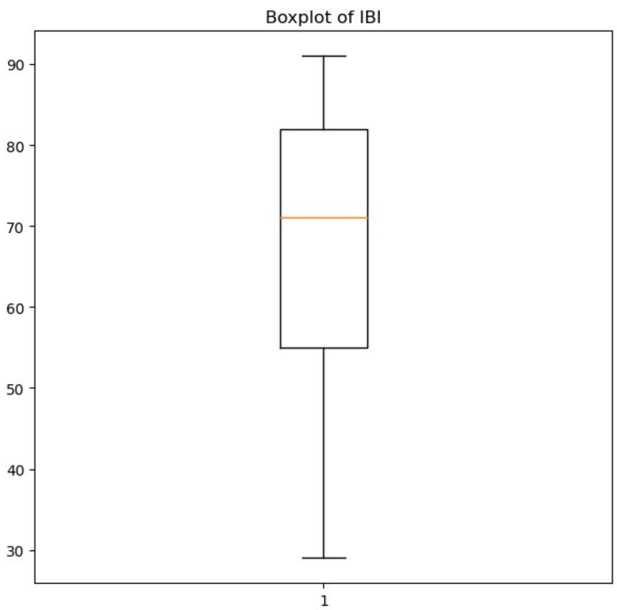
```
#Continue Problem 5

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Define the data
forest_area = np.array([0, 0, 0, 0, 0, 0, 3, 3, 7, 8, 9, 10, 10, 11,
14, 17, 17, 18, 21, 22,
25, 31, 32, 33, 33, 33, 39, 41, 43, 43, 47, 49, 49, 52, 52, 59, 63,
68, 75, 79, 79, 80, 86, 89, 90, 95, 95, 100, 100])
ibi = np.array([47, 61, 39, 59, 72, 76, 85, 89, 74, 89, 33, 46, 32,
80, 80, 78, 53, 43, 88, 84,
62, 55, 29, 29, 54, 78, 71, 55, 58, 71, 33, 59, 81, 71, 75, 64, 41,
82, 60, 84, 83, 82, 82, 86, 79, 67, 56, 85, 91])
```

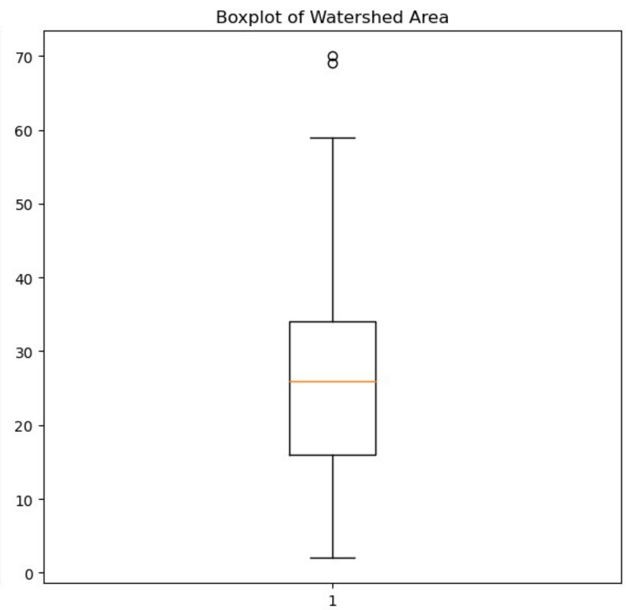
```
watershed_area = np.array([21, 29, 31, 32, 34, 34, 49, 52, 2, 70, 6,
28, 21, 59, 69, 47, 8, 8, 58, 54, 10, 57,
18, 19, 39, 49, 9, 5, 14, 9, 23, 31, 18, 16, 21, 32, 10, 26, 9, 54,
12, 21, 27, 23, 26, 16, 26, 26, 28])

# Convert to DataFrame
df = pd.DataFrame({'ForestArea': forest_area, 'IBI': ibi,
'WatershedArea': watershed_area})

# Filter the data where WatershedArea <= 70
df = df[df['WatershedArea'] <= 70]

# Calculate numerical summaries
print(df['IBI'].describe())
print(df['WatershedArea'].describe())

# Create histograms
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(df['IBI'], bins=10, color='blue', alpha=0.7)
plt.title('Histogram of IBI')
plt.xlabel('IBI')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(df['WatershedArea'], bins=10, color='green', alpha=0.7)
plt.title('Histogram of Watershed Area')
plt.xlabel('Watershed Area')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

# Create boxplots
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.boxplot(df['IBI'])
plt.title('Boxplot of IBI')

plt.subplot(1, 2, 2)
plt.boxplot(df['WatershedArea'])
plt.title('Boxplot of Watershed Area')

plt.tight_layout()
plt.show()

# Create a scatter plot of IBI vs Area
plt.figure(figsize=(8, 6))
plt.scatter(df['WatershedArea'], df['IBI'], color='blue', alpha=0.7)
plt.title('Scatter plot of IBI vs Watershed Area')
```

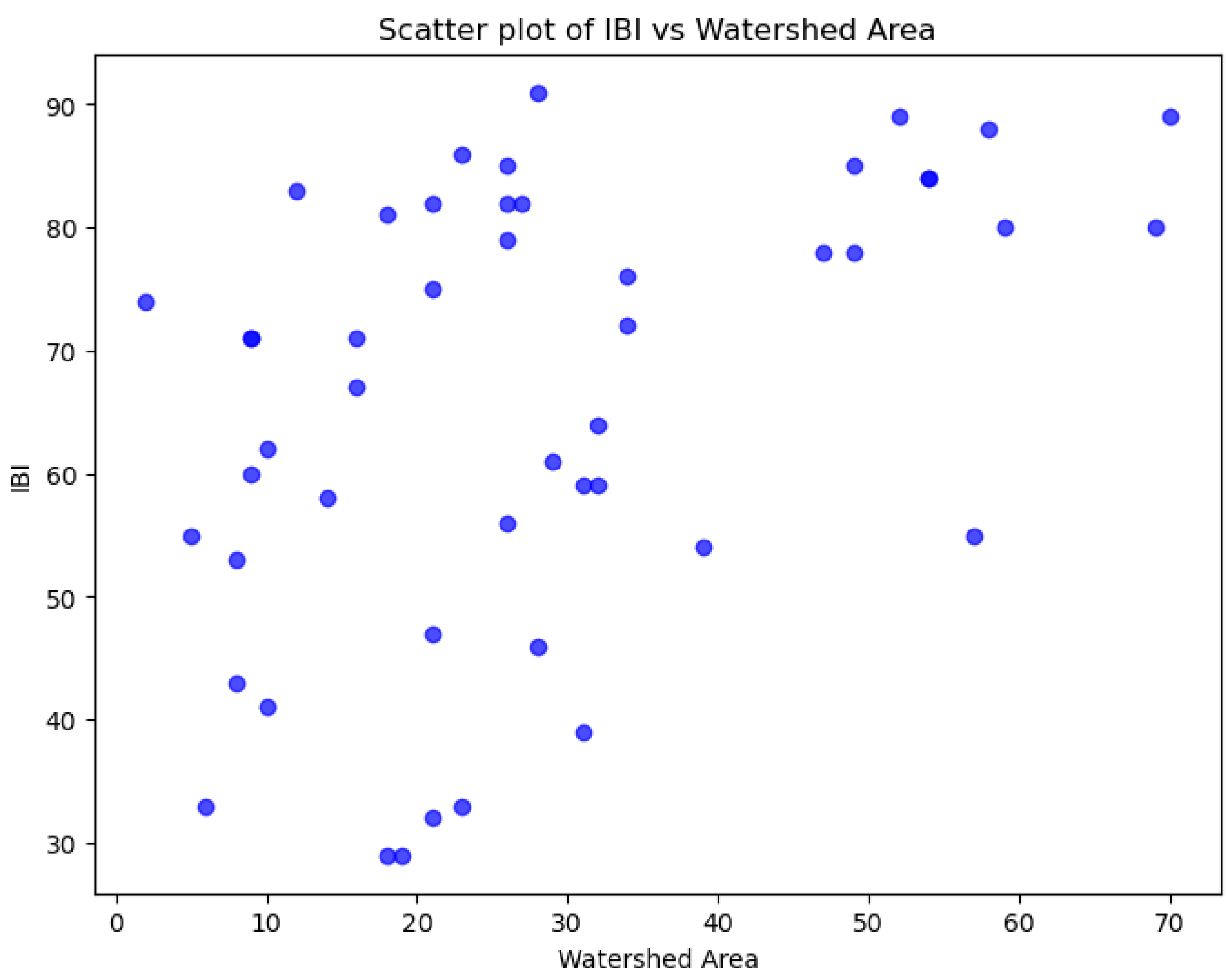
```
plt.xlabel('Watershed Area')
plt.ylabel('IBI')
plt.show()



#we can see from the box plot of watershed area that there are 2
outliers at 70
```

```
count    49.000000
mean     65.938776
std      18.279552
min      29.000000
25%      55.000000
50%      71.000000
75%      82.000000
max      91.000000
Name: IBI, dtype: float64
count    49.000000
mean     28.285714
std      17.714166
min       2.000000
25%      16.000000
50%      26.000000
75%      34.000000
max      70.000000
Name: WatershedArea, dtype: float64
```

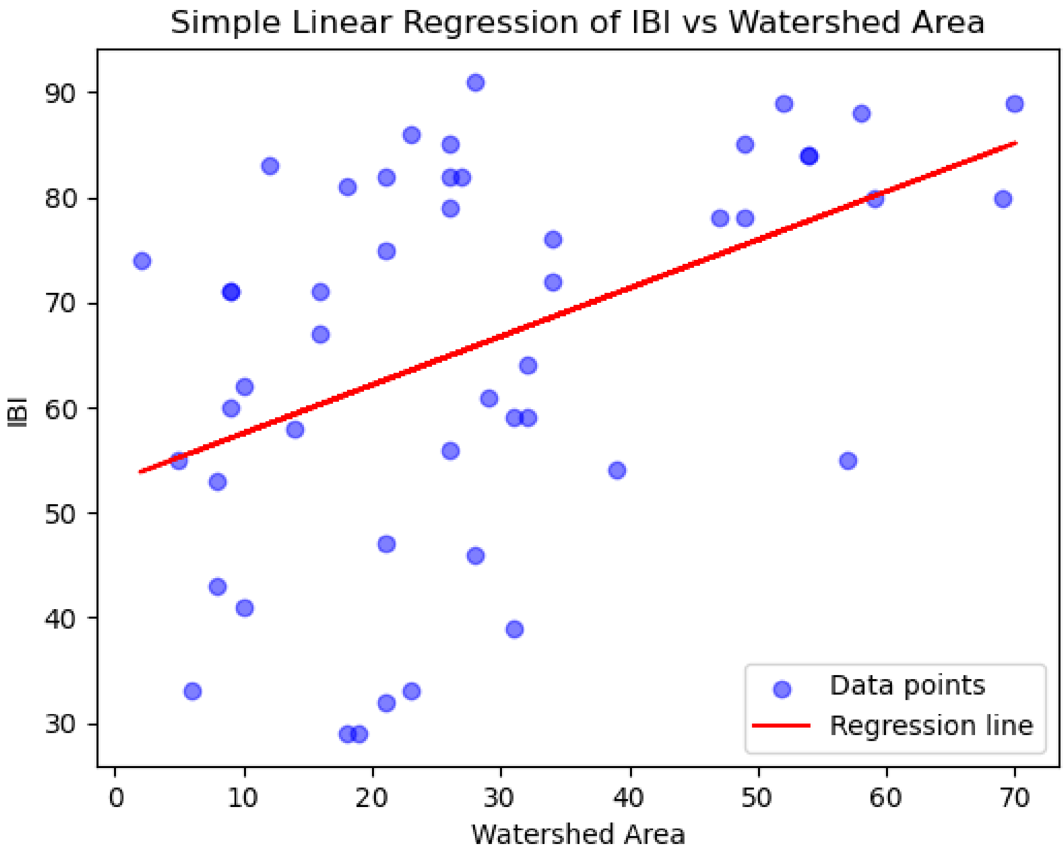


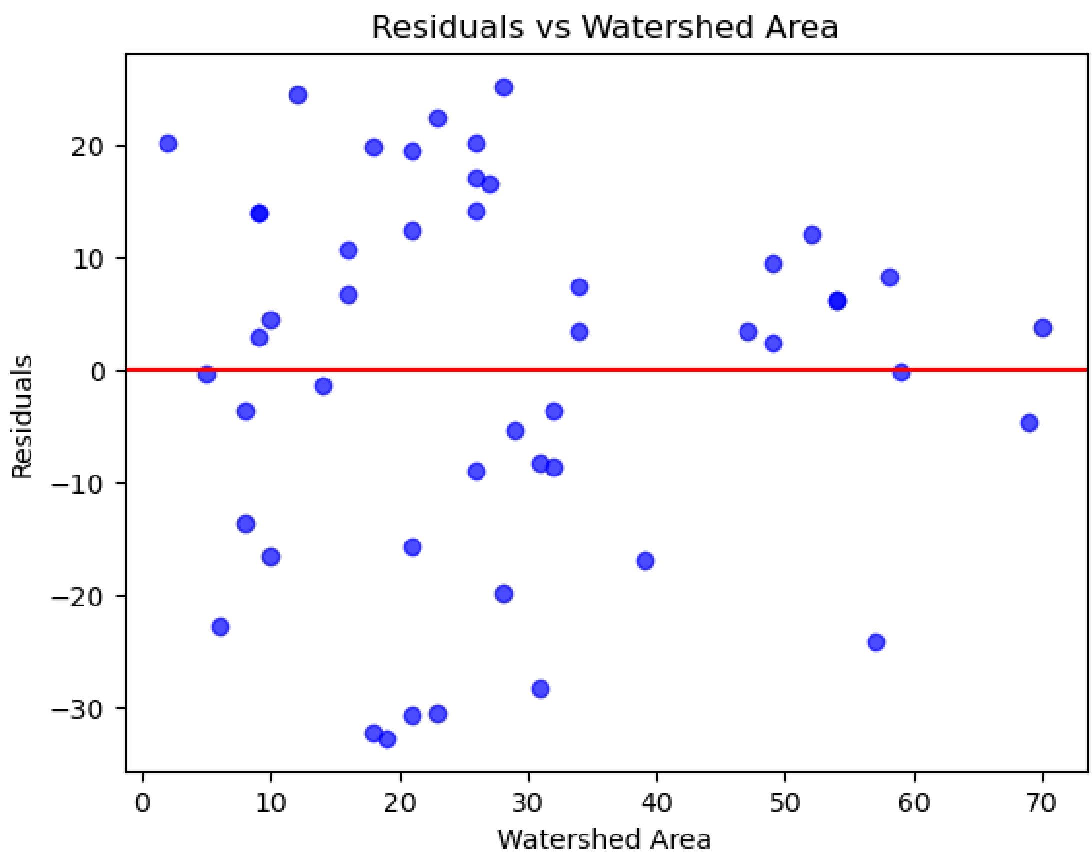

Boxplot of IBI
90
80
70
60
50
40
30
1


Boxplot of Watershed Area
70
60
50
40
30
20
10
0
1


Scatter plot of IBI vs Watershed Area
IBI
90
80
70
60
50
40
30
0
10
20
30
40
50
60
70
Watershed Area

```python
# Get the coefficients from the regression results
beta0 = results.params[0]
beta1 = results.params[1]

# Calculate the values of the dependent variable for the regression
line
y_pred = beta0 + beta1 * df['WatershedArea']

# Create a scatter plot of the data
plt.scatter(df['WatershedArea'], df['IBI'], color='blue', alpha=0.5,
label='Data points')

# Create a line plot of the regression line
plt.plot(df['WatershedArea'], y_pred, color='red', label='Regression
line')

# Add labels and title
plt.xlabel('Watershed Area')
plt.ylabel('IBI')
plt.title('Simple Linear Regression of IBI vs Watershed Area')
plt.legend()

# Show the plot
plt.show()

# Calculate the residuals
residuals = df['IBI'] - y_pred

# Create a scatter plot of the residuals versus area
plt.scatter(df['WatershedArea'], residuals, color='blue', alpha=0.7)
plt.axhline(0, color='red')  # This adds a horizontal line at y=0 for
reference
plt.title('Residuals vs Watershed Area')
plt.xlabel('Watershed Area')
plt.ylabel('Residuals')
plt.show()
```


Simple Linear Regression of IBI vs Watershed Area
IBI
Watershed Area
Data points
Regression line
90
80
70
60
50
40
30
0
10
20
30
40
50
60
70


Residuals vs Watershed Area
Residuals
Watershed Area
20
10
0
−10
−20
−30
0
10
20
30
40
50
60
70