ENGS 93 HW 4

Cameron Wolfe 2/28/2024

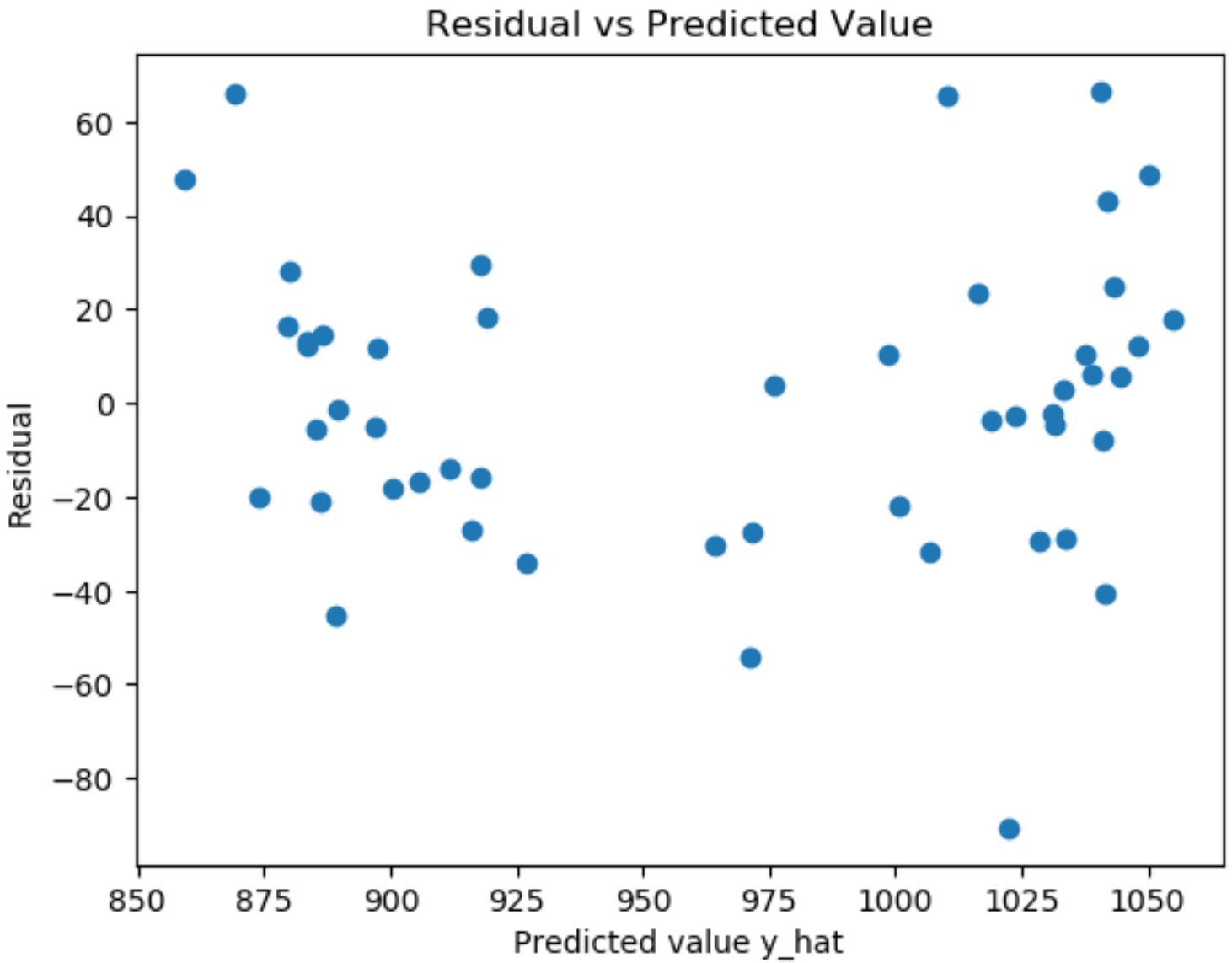
In [ ]:
```python
import numpy as np
from scipy.stats import t, norm, chi2, f
from math import sqrt, exp, ceil, floor
import matplotlib.pyplot as plt

def normal_probability_plot(
    data_array: np.ndarray,
    title: str = "Normal Probability Plot",
) -> None:
    sorted_array = np.sort(data_array.flatten())

    n = len(sorted_array)

    z = np.zeros(n)

    for i in range(n):
        z[i] = norm.ppf((i + 0.5) / n)

    plt.scatter(sorted_array, z)
    plt.title(title)
    plt.xlabel("Values")
    plt.ylabel("Z value")
    plt.show()
```

Problem 1

In [ ]:
```
sat_data = np.genfromtxt("sat.csv", delimiter=",")[1:, 1:]
```

Part a

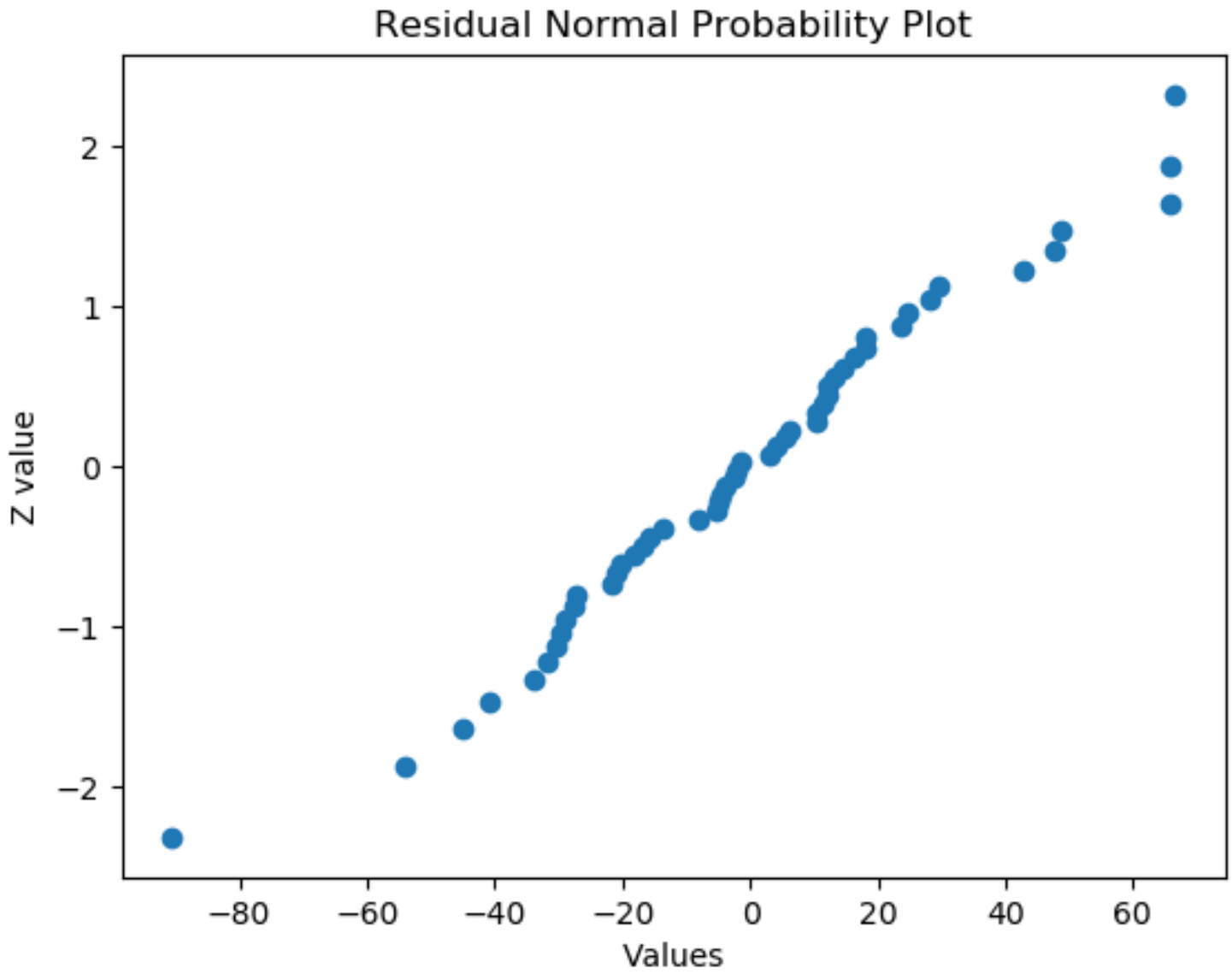
In [ ]:
```
x_data = sat_data[:, 0:4]
y = sat_data[:, 6]
num_data_points = len(y)
x = np.concatenate([np.ones((num_data_points, 1)), x_data], axis=1)
beta = np.linalg.solve(np.matmul(np.transpose(x), x), np.matmul(np.transpose(x), y))

y_hat = np.matmul(x, beta)
residual = y - y_hat

plt.scatter(y_hat, residual)
plt.title("Residual vs Predicted Value")
plt.xlabel("Predicted value y_hat")
plt.ylabel("Residual")
plt.show()
```



Variance appears to remain constant

Part b

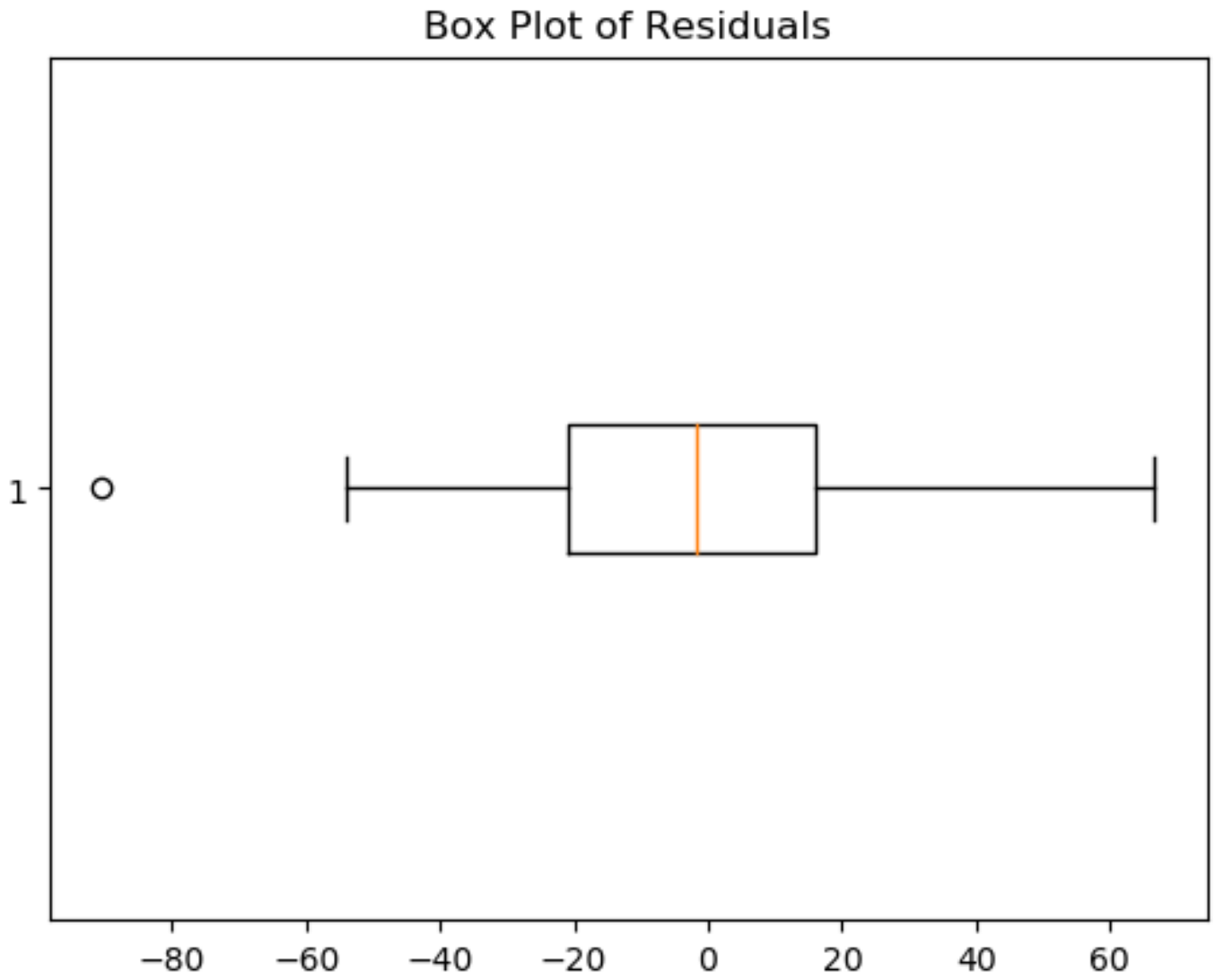
In [ ]:
```
normal_probability_plot(
    np.reshape(residual, (-1)),
    "Residual Normal Probability Plot"
)
```



Residuals appear to be normally distributed

Part c

In [ ]:
```
plt.boxplot(residual, vert=False)
plt.title("Box Plot of Residuals")
plt.show()
```



As we can see that there is an outlier, as it is outside the whiskers on the box plot. The outlier state is West Virginia.

Problem 2

In [ ]:
```
beta = np.array([300, 0.85, 10.40])
SST = 1230.5
SSE = 120.3
alpha = 0.05
n = 15
k = 2
p = 3
```

Part a

In [ ]:
```
SSR = SST - SSE
F_0 = SSR / k / (SSE / (n - p))
f_alpha = f.ppf(1-alpha, k, n - p)

print("Test Statistic:", F_0)
print("Critical Value:", f_alpha)
```

```
Test Statistic: 55.3715710723192
Critical Value: 3.8852938346523933
```

Can reject the null hypothesis as the test statistic is greater than the critical value. Therefore we can declare that the regression is significant.

Part b

In [ ]:
```
R_squared = SSR / SST
adjusted_R_squared = 1 - SSE / (n - p) / (SST / (n - 1))
```

```
print("R^2 value:", R_squared)
print("Adjusted R^2 value:", adjusted_R_squared)
```

```
R^2 value: 0.902234863876473
Adjusted R^2 value: 0.8859406745225518
```

~90% of the variance in the data can be explained by the regression, depending on whether you use the standard or adjusted R^2 statistic.

Part c

In [ ]:
```
mse = SSE / (n - p)
print("Old MSE:", mse)
new_SSE = 117.20
k = k + 1
p = p + 1
new_mse = new_SSE / (n - p)
print("New MSE:", new_mse)
```

```
Old MSE: 10.025
New MSE: 10.654545454545454
```

As can be seen, the MSE increased from the original regression, meaning that by adding the new parameter, we are overfitting. This can also be seen by looking at the adjusted R^2 value.

In [ ]:
```
R_squared = 1 - new_SSE / SST
adjusted_R_squared = 1 - new_SSE / (n - p) / (SST / (n - 1))
print("R^2 value:", R_squared)
print("Adjusted R^2 value:", adjusted_R_squared)
```

```
R^2 value: 0.904754164973588
Adjusted R^2 value: 0.8787780281482028
```

While the R^2 increases, the adjusted R^2 decreases, meaning that by adding the new parameter, we are overfitting the data.

Part d

In [ ]:
```
SSR_b3_given_b2 = (SST - new_SSE) - (SST - SSE)
F_0 = SSR_b3_given_b2 / 1 / (new_SSE / (n - p))
f_alpha = f.ppf(1-alpha, k, n - p)

print("Test statistic:", F_0)
print("Critical value:", f_alpha)
```

```
Test statistic: 0.29095563139930886
Critical value: 3.5874337024204936
```

Cannot reject H_0 as the test statistic is less than the critical value. This means we can declare that the new variable addition is not significant.

Problem 3

Part a

In [ ]:
```
heat_cement_data = np.genfromtxt("heat_cement.csv", delimiter=",")[1:,:]
y = heat_cement_data[:, 0]
x_data = heat_cement_data[:, 1:]
```

```
num_data_points = len(y)
p = 4
x = np.concatenate([np.ones((num_data_points, 1)), x_data], axis=1)
beta = np.linalg.solve(np.matmul(np.transpose(x), x), np.matmul(np.transpose(x), y))
print("Regression parameters:", beta)
print("y_hat = " + str(beta[0]) + " + " + str(beta[1]) + " * x_1 + " + str(beta[2]) +
```

```
Regression parameters: [-39.91967442   0.7156402    1.29528612  -0.15212252]
y_hat = -39.91967442012454 + 0.7156402004852787 * x_1 + 1.2952861243885896 * x_2 + -0.1
5212251914864716 * x_3
```

Part b

In [ ]:

```
residual = y - np.matmul(x, beta)
SST = np.sum(y ** 2)
SSE = np.sum(residual ** 2)

R_squared = 1 - SSE / SST
print("R^2:", R_squared)
```

```
R^2: 0.9790056396339095
```

Part c

In [ ]:

```
new_x = x[:, [0, 2, 3]]
new_beta = np.linalg.solve(
    np.matmul(np.transpose(new_x), new_x),
    np.matmul(np.transpose(new_x), y)
)
r = 1
new_residual = y - np.matmul(new_x, new_beta)
```

```
new_SSE = np.sum(new_residual ** 2)

SSR_b1_given_b3 = new_SSE - SSE

F_0 = SSR_b1_given_b3 / r / (SSE / (n - p))
f_alpha = f.ppf(1-alpha, r, n-p)

print("Test statistic:", F_0)
print("Critical value:", f_alpha)
```

```
Test statistic: 18.221268096922067
Critical value: 4.844335674943618
```

Can conclude that % of tricalcium aluminate is significant in explaining the heat evolved during the hardening of cement.

Problem 4

Part a

In [ ]:
```
airline_data = np.genfromtxt("airline.csv", delimiter=",")[1:, :]
y = airline_data[:, 0]
x_data = airline_data[:, 1:]
n = len(y)
p = 3
k = p + 1
x = np.concatenate([np.ones((n, 1)), x_data], axis=1)
beta = np.linalg.solve(np.matmul(np.transpose(x), x), np.matmul(np.transpose(x), y))
print("Regression coefficients:", beta)
```

```
Regression coefficients: [  2.53732669   0.91930855 659.26822475   2.17963956]
```

Part b

In [ ]:
```python
residual = y - np.matmul(x, beta)
SSE = np.sum(residual ** 2)
SST = np.sum(y ** 2)
SSR = SST - SSE

F0 = (SSR / k) / (SSE / (n - p))
p = f.sf(F0, k, n - p)
print("P-value:", p)
```

```
P-value: 2.790106890568091e-34
```

Because $p < \alpha = 0.01$, can reject H_0 and conclude that there is a significant relationship between the parameters and the data.

Part c

In [ ]:
```python
new_x = x[:, 0:2]
new_beta = np.linalg.solve(
    np.matmul(np.transpose(new_x), new_x),
    np.matmul(np.transpose(new_x), y)
)
new_residual = y - np.matmul(new_x, new_beta)
new_SSE = np.sum(new_residual ** 2)
new_SSR = SST - new_SSE
SSR_b2b3_given_b1 = SSR - new_SSR
r = 2
```

```
p = 4
T0 = SSR_b2b3_given_b1 / r / (SSE / (n - p))
p = f.sf(T0, r, n-p)
print("P-value:", p)
```

```
P-value: 8.810318249346287e-15
```

Because $p < \alpha = 0.01$, we can reject H_0 and conclude that β_2 and β_3 add significant prediction power to the regression

Problem 5

In [ ]:
```
sample_means = np.array([5.43, 3.99, 4.86, 5.65, 4.80])
sample_variances = np.array([2.66, 3.08, 2.57, 3.38, 2.60])
I = 5
measurements = 40 * np.ones(I)
N = np.sum(measurements)
total_mean = np.sum(sample_means * measurements) / N
```

Part a

In [ ]:
```
SST = (
    np.sum(sample_variances * (measurements - 1) + measurements * sample_means **2)
     - N * total_mean ** 2
)

SS_treatment = np.sum(measurements * sample_means ** 2) - N * total_mean**2
SSE = SST - SS_treatment
```

```
MSE = SSE / (N - 1)
print("Within group variance:", MSE)
```

```
Within group variance: 2.800552763819093
```

Part b

In [ ]:
```
F_0 = (SS_treatment / (I - 1)) / (SSE / (N - 1))
print("F-statistic:", F_0)
```

```
F-statistic: 5.972106726956227
```

Part c

In [ ]:
```
p = f.sf(F_0, I - 1, N - 1)
print("P-value:", p)
```

```
P-value: 0.00014723911102014129
```

Using $\alpha = 0.05$, we can conclude that there are two populations that are significantly different as $p < \alpha$

Problem 6

In [ ]:
```
fatigue_crack_data = np.array([[2.29, 2.06, 1.90],
[2.47, 2.05, 1.93],
[2.48, 2.23, 1.75],
[2.12, 2.03, 2.06],
[2.65, 3.20, 3.10],
```

```
[2.68, 3.18, 3.24],
[2.06, 3.96, 3.98],
[2.38, 3.64, 3.24],
[2.24, 11.00, 9.96],
[2.71, 11.00, 10.01],
[2.81, 9.06, 9.36],
[2.08, 11.3, 10.40]])
I, J, K = 3, 3, 4

fatigue_crack_data = np.transpose(np.reshape(np.transpose(fatigue_crack_data), (I, J,
```

Part a

In [ ]:

```
treatment_means = np.mean(fatigue_crack_data, 2)
row_means = np.mean(treatment_means, 1)
column_means = np.mean(treatment_means, 0)
grand_mean = np.mean(row_means)

row_effects = row_means - grand_mean
column_effects = column_means - grand_mean
interaction_effects = np.zeros((3,3))
for i in range(I):
    for j in range(J):
        interaction_effects[i, j] = (
            treatment_means[i, j]
            - row_means[i]
            - column_means[j]
            + grand_mean
            )

error = np.zeros(fatigue_crack_data.shape)
```

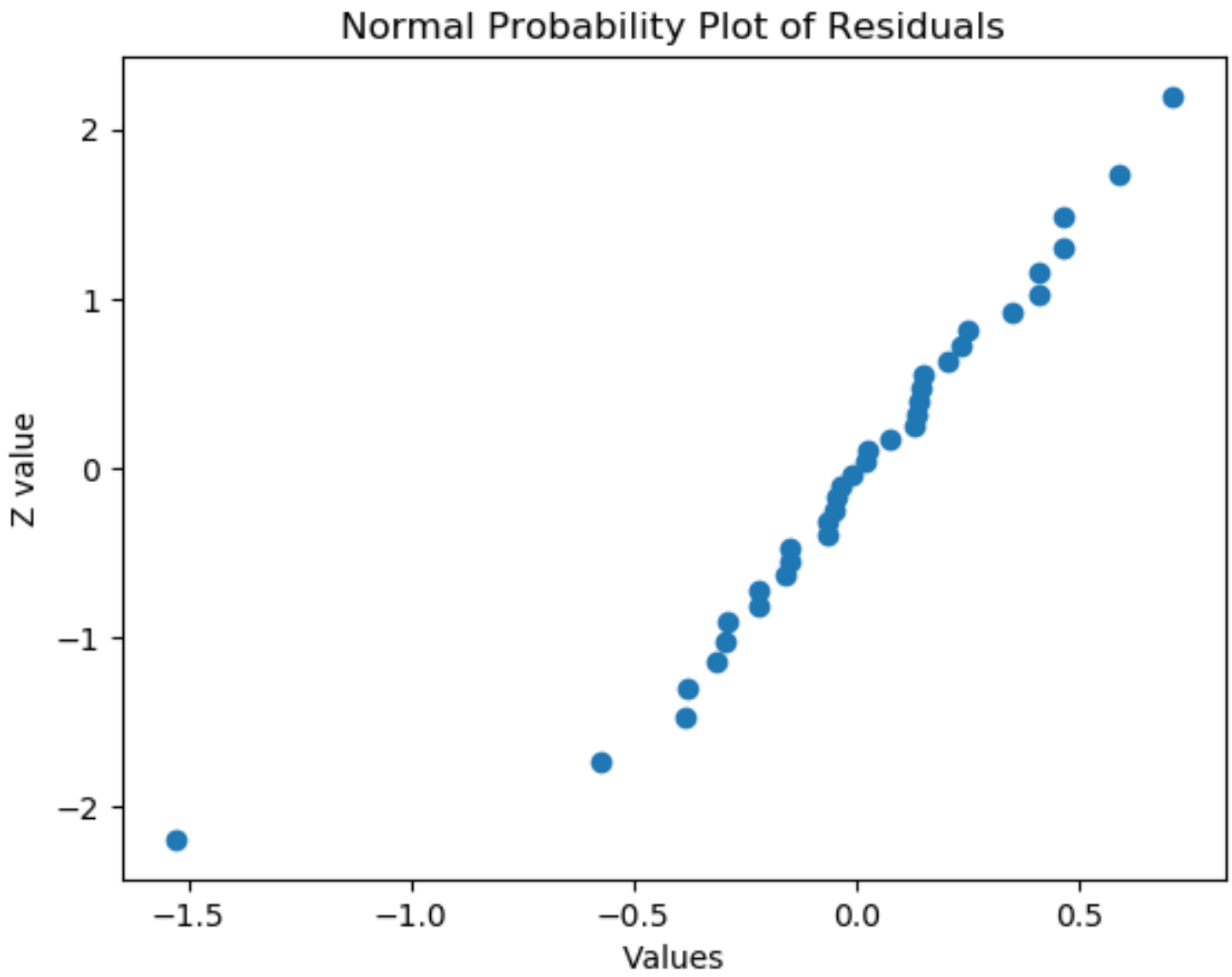
```
for i in range(I):
    for j in range(J):
        for k in range(K):
            error[i,j,k] = fatigue_crack_data[i,j,k] - treatment_means[i,j]

SSE = np.sum(np.sum(np.sum(error ** 2)))
SSAB = K * np.sum(np.sum(interaction_effects ** 2))
SSA = J * K * np.sum(row_effects)
SSB = I * K * np.sum(column_effects)

MSE = SSE / (I * J * (K - 1))
MSAB = SSAB / ((I - 1) * (J - 1))
MSA = SSA / (I - 1)
MSB = SSB / (J - 1)

F_AB = MSAB / MSE
F_A = MSA / MSE
F_B = MSB / MSE

p_AB = f.sf(F_AB, (I - 1) * (J - 1), I * J * (K - 1))
p_A = f.sf(F_A, (I - 1), I * J * (K - 1))
p_B = f.sf(F_B, (J - 1), I * J * (K - 1))

print("Interaction P-value:", p_AB)
print("Row Factor P-value:", p_A)
print("Column Factor P-value:", p_B)
```

```
Interaction P-value: 4.32406971826135e-17
Row Factor P-value: 0.999999999999988
Column Factor P-value: 0.999999999999973
```

From this, we can reject the additive model, meaning that it is the combination of factors that causes changes in crack growth rate, and not just frequency or environment.

Part b

In [ ]:
```
normal_probability_plot(error.reshape((-1)), "Normal Probability Plot of Residuals")
```

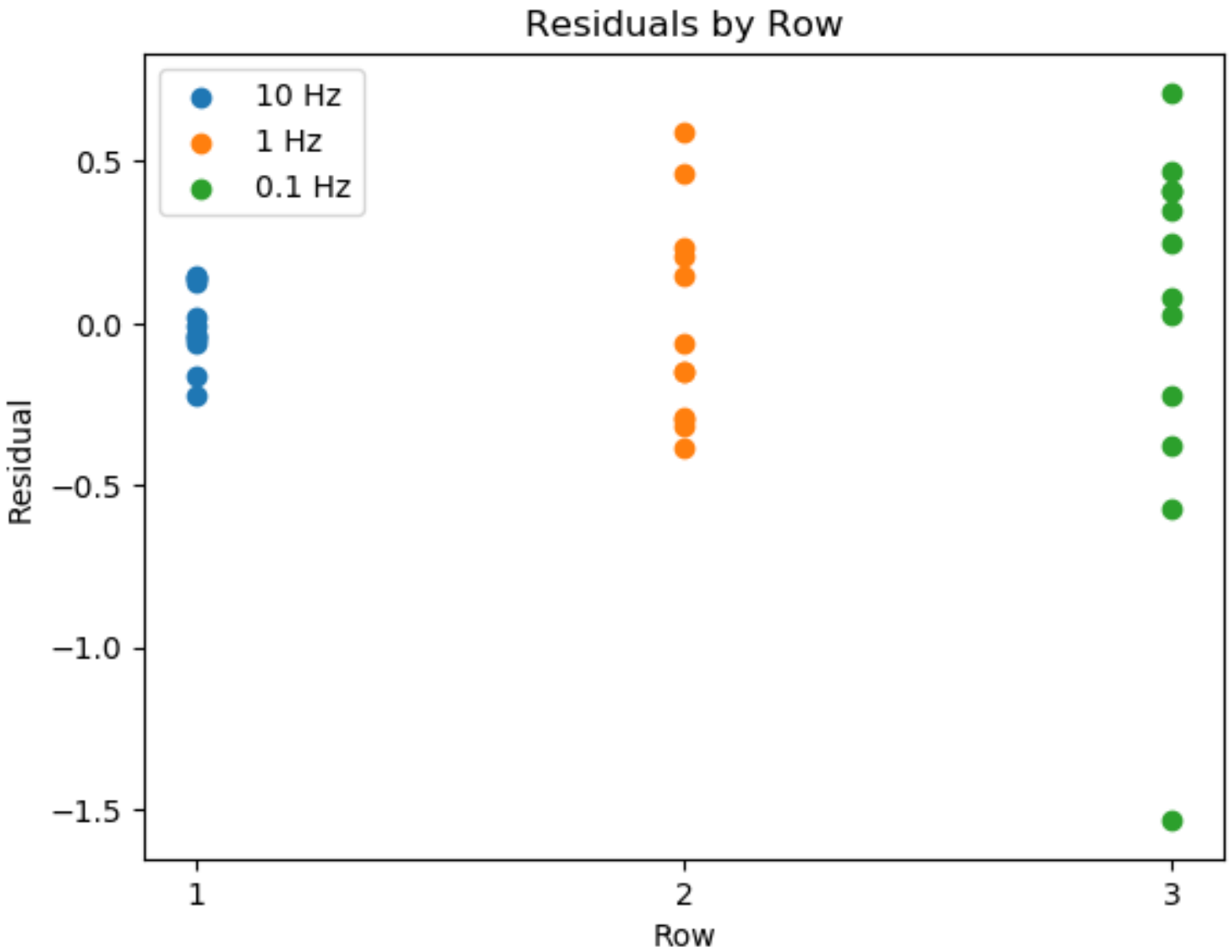


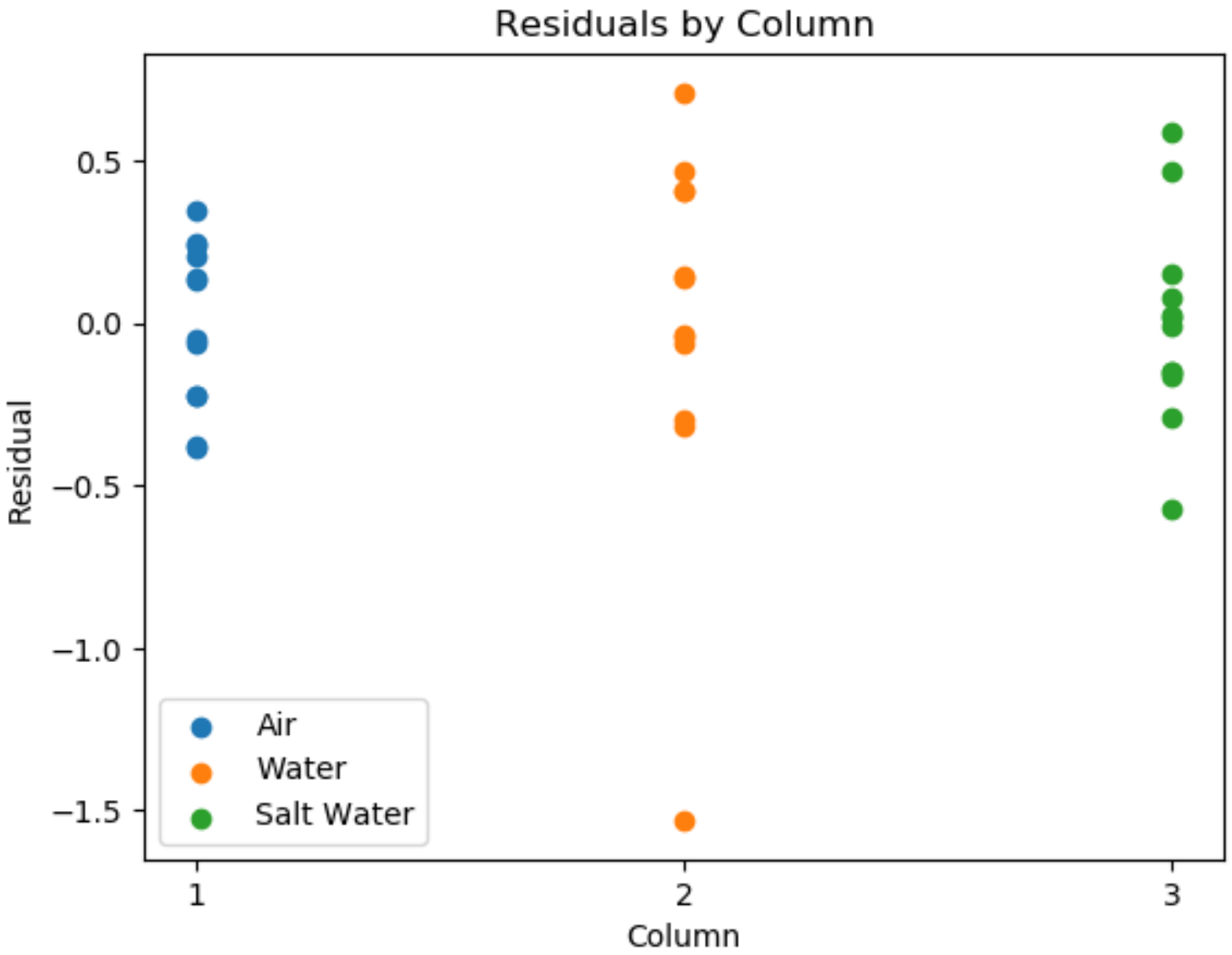
Residuals appear to be normally distributed, with one outlier.

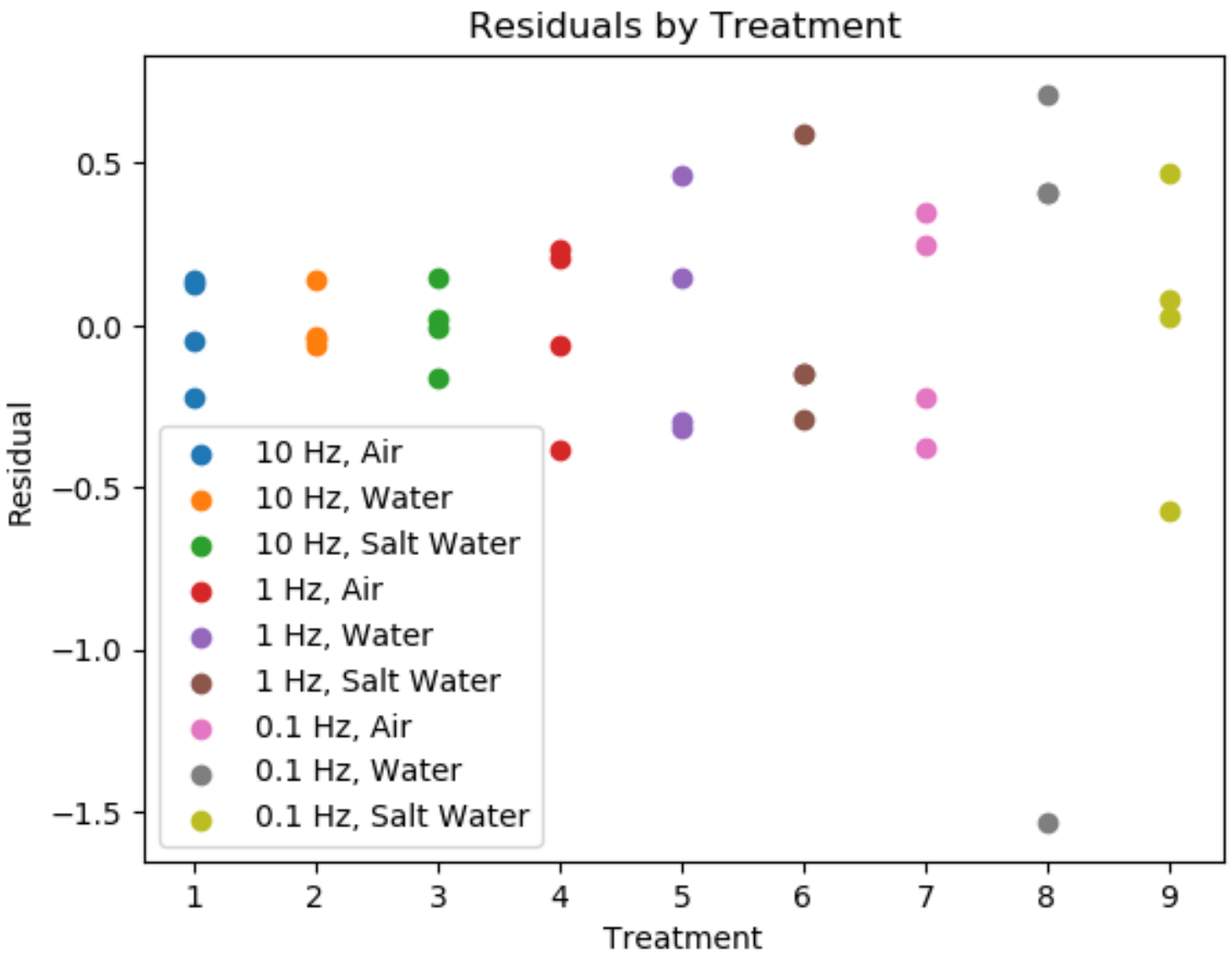
In [ ]:
```
frequency_values = ["10 Hz", "1 Hz", "0.1 Hz"]
environment_types = ["Air", "Water", "Salt Water"]
```

```
residuals_by_row = []
residuals_by_column = []
residuals_by_treatment = []
for i in range(I):
    residuals_by_row.append(error[i, :, :].reshape((-1)))
    label_x = np.full((J*K), str(i+1), dtype=str)
    plt.scatter(label_x, residuals_by_row[i], label=frequency_values[i])
plt.title("Residuals by Row")
plt.xlabel("Row")
plt.ylabel("Residual")
plt.legend(loc="best")
plt.show()

for j in range(J):
    residuals_by_column.append(error[:, j, :].reshape((-1)))
    label_x = np.full((I*K), str(j+1), dtype=str)
    plt.scatter(label_x, residuals_by_column[j], label=environment_types[j])
plt.title("Residuals by Column")
plt.xlabel("Column")
plt.ylabel("Residual")
plt.legend(loc="best")
plt.show()

for i in range(I):
    for j in range(J):
        residuals_by_treatment.append(error[i,j,:].reshape((-1)))
        label_x = np.full((K), str(3 * i + j + 1), dtype=str)
        plt.scatter(label_x, residuals_by_treatment[-1], label=(frequency_values[i] +
plt.title("Residuals by Treatment")
plt.xlabel("Treatment")
plt.ylabel("Residual")
```

```
plt.legend(loc="best")
plt.show()
```




Residuals by Column
Residual
0.5
0.0
−0.5
−1.0
−1.5
1
2
3
Column
Air
Water
Salt Water



Residuals appear to have consistent variance based on row, column, and treatment, with the exception of what appears to be an outlier in the 0.1 Hz, Water treatment.

Part c

In [ ]:

```
fatigue_crack_data = np.log(fatigue_crack_data)
treatment_means = np.mean(fatigue_crack_data, 2)
row_means = np.mean(treatment_means, 1)
column_means = np.mean(treatment_means, 0)
grand_mean = np.mean(row_means)

row_effects = row_means - grand_mean
column_effects = column_means - grand_mean
interaction_effects = np.zeros((3,3))
for i in range(I):
    for j in range(J):
        interaction_effects[i, j] = treatment_means[i, j] - row_means[i] - column_mean

error = np.zeros(fatigue_crack_data.shape)
for i in range(I):
    for j in range(J):
        for k in range(K):
            error[i,j,k] = fatigue_crack_data[i,j,k] - treatment_means[i,j]

SSE = np.sum(np.sum(np.sum(error ** 2)))
SSAB = K * np.sum(np.sum(interaction_effects ** 2))
SSA = J * K * np.sum(row_effects)
SSB = I * K * np.sum(column_effects)

MSE = SSE / (I * J * (K - 1))
MSAB = SSAB / ((I - 1) * (J - 1))
MSA = SSA / (I - 1)
MSB = SSB / (J - 1)
```

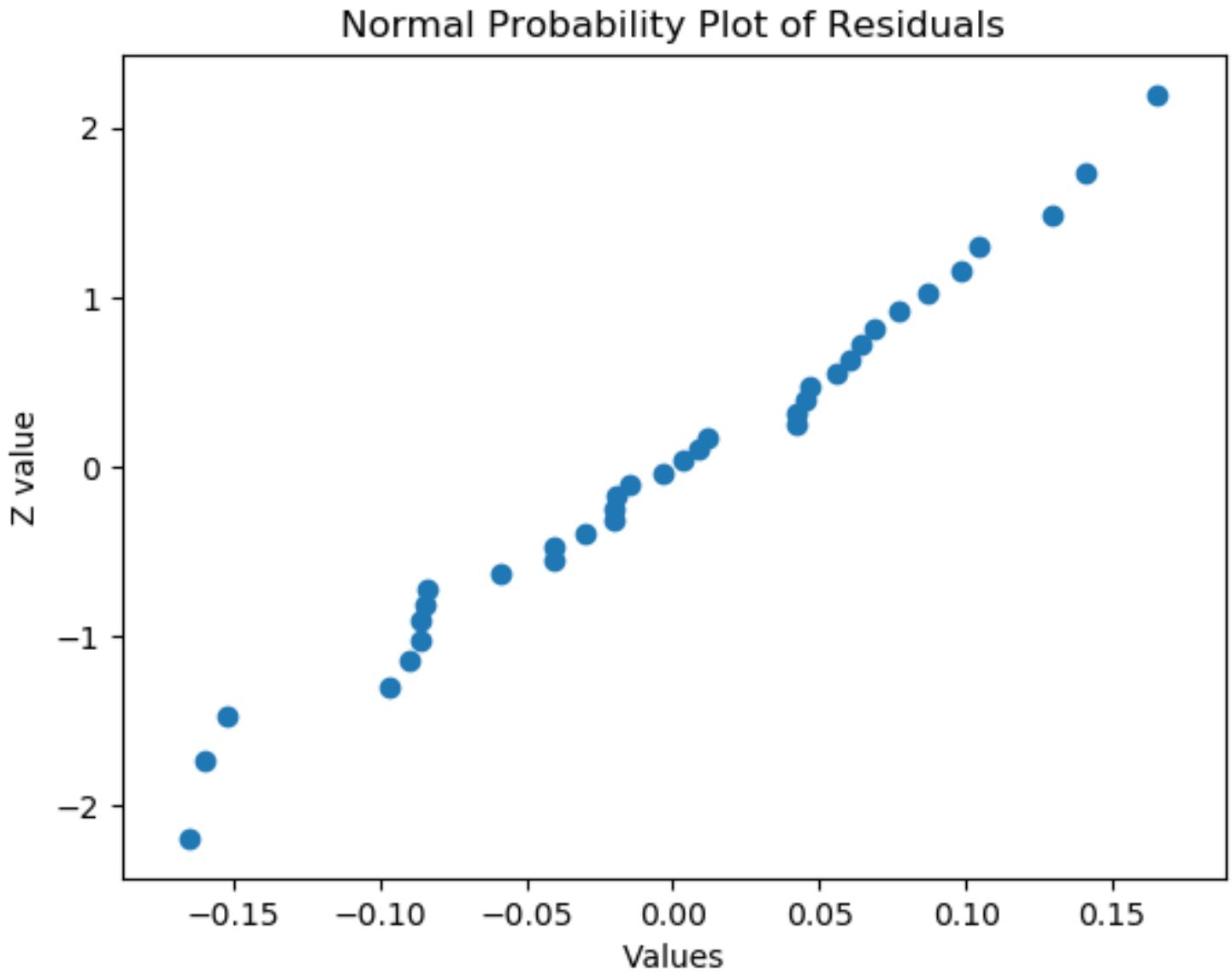
```
F_AB = MSAB / MSE
F_A = MSA / MSE
F_B = MSB / MSE

p_AB = f.sf(F_AB, (I - 1) * (J - 1), I * J * (K - 1))
p_A = f.sf(F_A, (I - 1), I * J * (K - 1))
p_B = f.sf(F_B, (J - 1), I * J * (K - 1))

print("Interaction P-value:", p_AB)
print("Row Factor P-value:", p_A)
print("Column Factor P-value:", p_B)
```

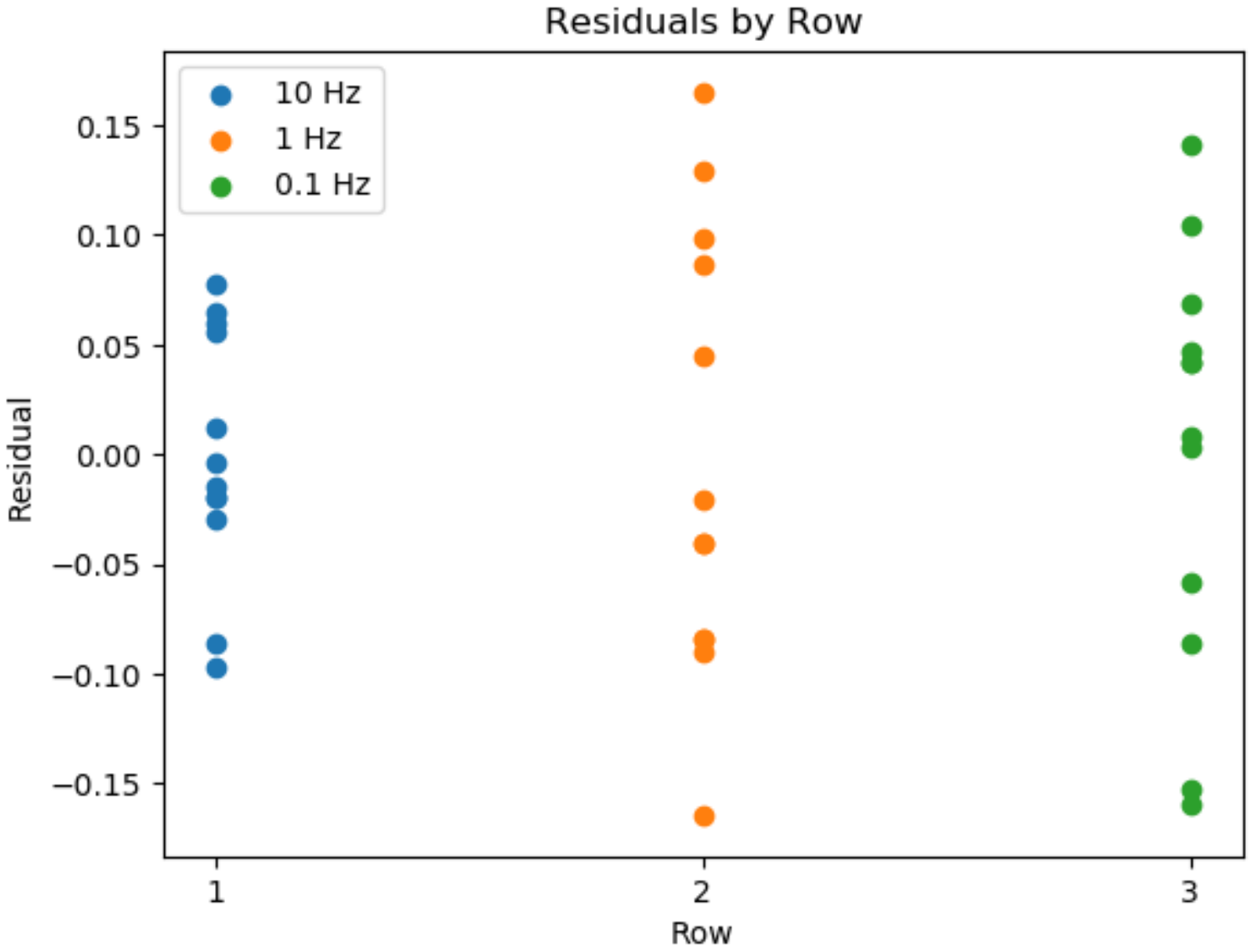
```
Interaction P-value: 1.8846726538688147e-15
Row Factor P-value: 1.0
Column Factor P-value: 1.0
```

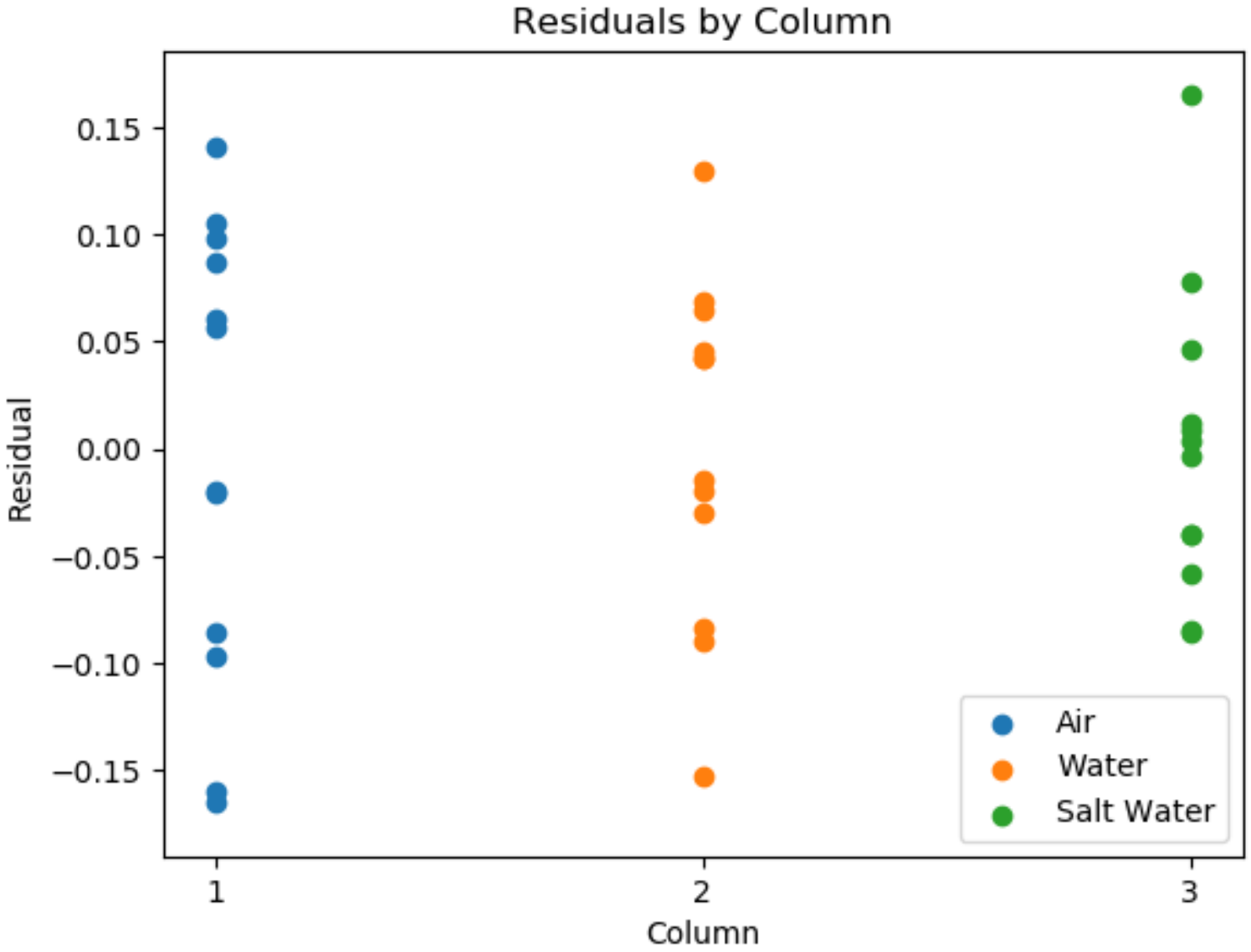
From this, we can again conclude that there is a significant interaction component as $p < 0.05$.

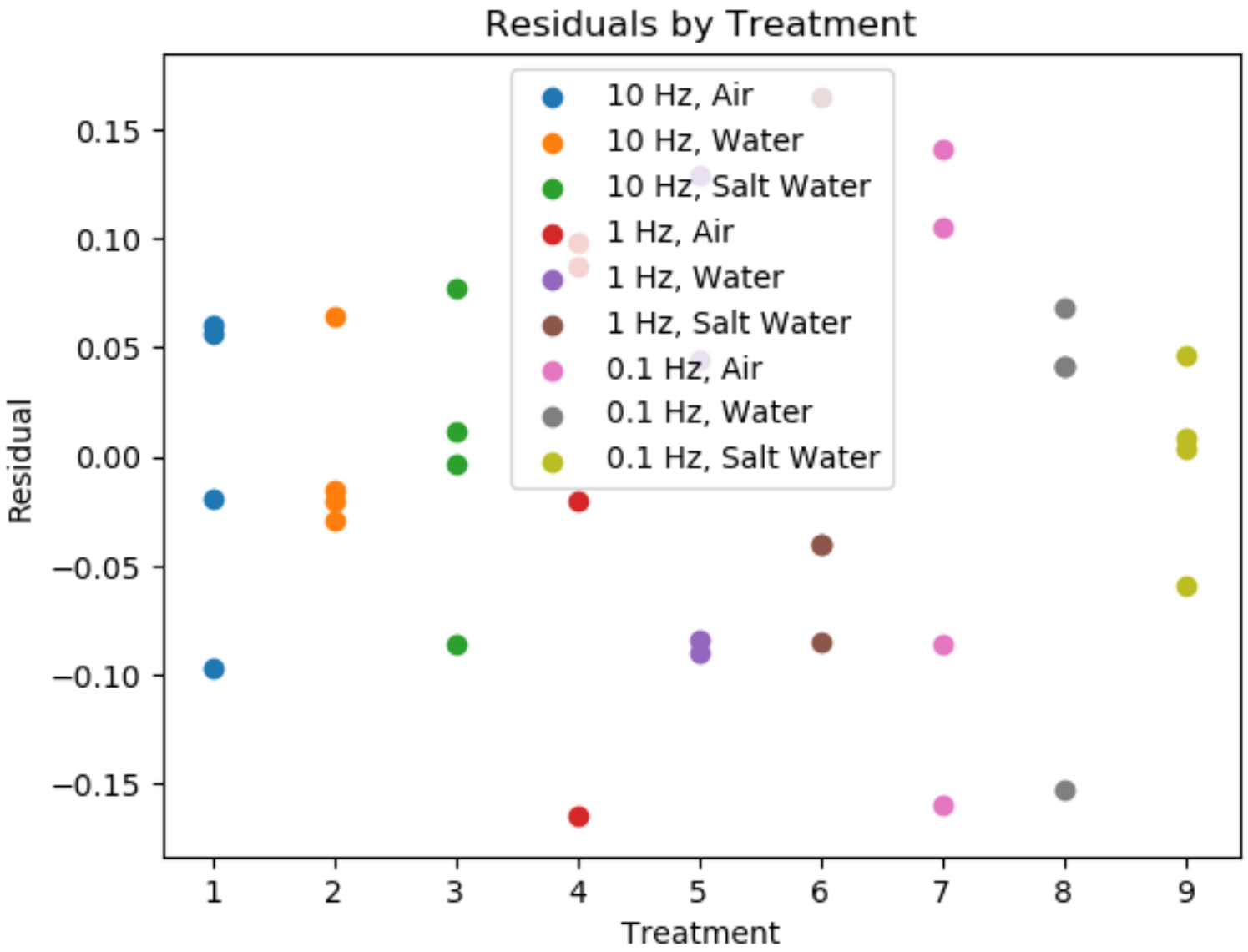
In [ ]:
```
normal_probability_plot(error.reshape((-1)), "Normal Probability Plot of Residuals")
frequency_values = ["10 Hz", "1 Hz", "0.1 Hz"]
environment_types = ["Air", "Water", "Salt Water"]
residuals_by_row = []
residuals_by_column = []
residuals_by_treatment = []
for i in range(I):
    residuals_by_row.append(error[i, :, :].reshape((-1)))
    label_x = np.full((J*K), str(i+1), dtype=str)
    plt.scatter(label_x, residuals_by_row[i], label=frequency_values[i])
plt.title("Residuals by Row")
plt.xlabel("Row")
plt.ylabel("Residual")
```

```
plt.legend(loc="best")
plt.show()

for j in range(J):
    residuals_by_column.append(error[:, j, :].reshape((-1)))
    label_x = np.full((I*K), str(j+1), dtype=str)
    plt.scatter(label_x, residuals_by_column[j], label=environment_types[j])
plt.title("Residuals by Column")
plt.xlabel("Column")
plt.ylabel("Residual")
plt.legend(loc="best")
plt.show()

for i in range(I):
    for j in range(J):
        residuals_by_treatment.append(error[i,j,:].reshape((-1)))
        label_x = np.full((K), str(3 * i + j + 1), dtype=str)
        plt.scatter(
            label_x, residuals_by_treatment[-1],
            label=(frequency_values[i] + ", " + environment_types[j])
        )
plt.title("Residuals by Treatment")
plt.xlabel("Treatment")
plt.ylabel("Residual")
plt.legend(loc="best")
plt.show()
```


Normal Probability Plot of Residuals
Z value
Values
2
1
0
−1
−2
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15


Residuals by Row
10 Hz
1 Hz
0.1 Hz
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
Residual
1
2
3
Row


Residuals by Column
Residual
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
1
2
3
Column
Air
Water
Salt Water



Taking the natural log of the data appears to make the residuals more normal in their distribution

Problem 7

Part a

First start with interaction γ:

Because $p_\gamma > 0.05$, we cannot conclude that there is interaction between the catalyst and reagent. This means that the model is additive.

Because the model is additive, we can then look at the row effects α, and the column effects β:

Reject that row effects (the catalyst used) are zero as $p_\alpha < 0.05$. This means that catalyst used has a signifcant impact on yield. Reject that column effects (the reagent used) are zero as $p_\beta < 0.05$. This means that reagent used has a signifcant impact on yield.

Part b

In [ ]:
```
SSA = 50
SSB = 100
SSAB = 2
SSE = 400
I, J, K = 2, 4, 8
N = I * J * K
SST = SSA + SSB + SSAB + SSE
```

```
population_variance = SST / (N - 1)
print("Population variance:", population_variance)
```

```
Population variance: 8.761904761904763
```

Part c

Since column factor is irrelevant, this means that SSE is comprised of the original SSE, SSB, and SSAB

In [ ]:
```
new_SSE = SSE + SSB + SSAB
new_MSE = new_SSE / (N - I)
F = (SSA / (I - 1)) / (new_MSE)
p = f.sf(F, I - 1, N - I)
print("P-value:", p)
```

```
P-value: 0.01566702009762702
```

Problem 8

In [ ]:
```
copper_data = np.genfromtxt("copper_content.csv", delimiter=",")[1:,1:]
n = 3
num_measurements = len(copper_data[:,0])
```

Part a

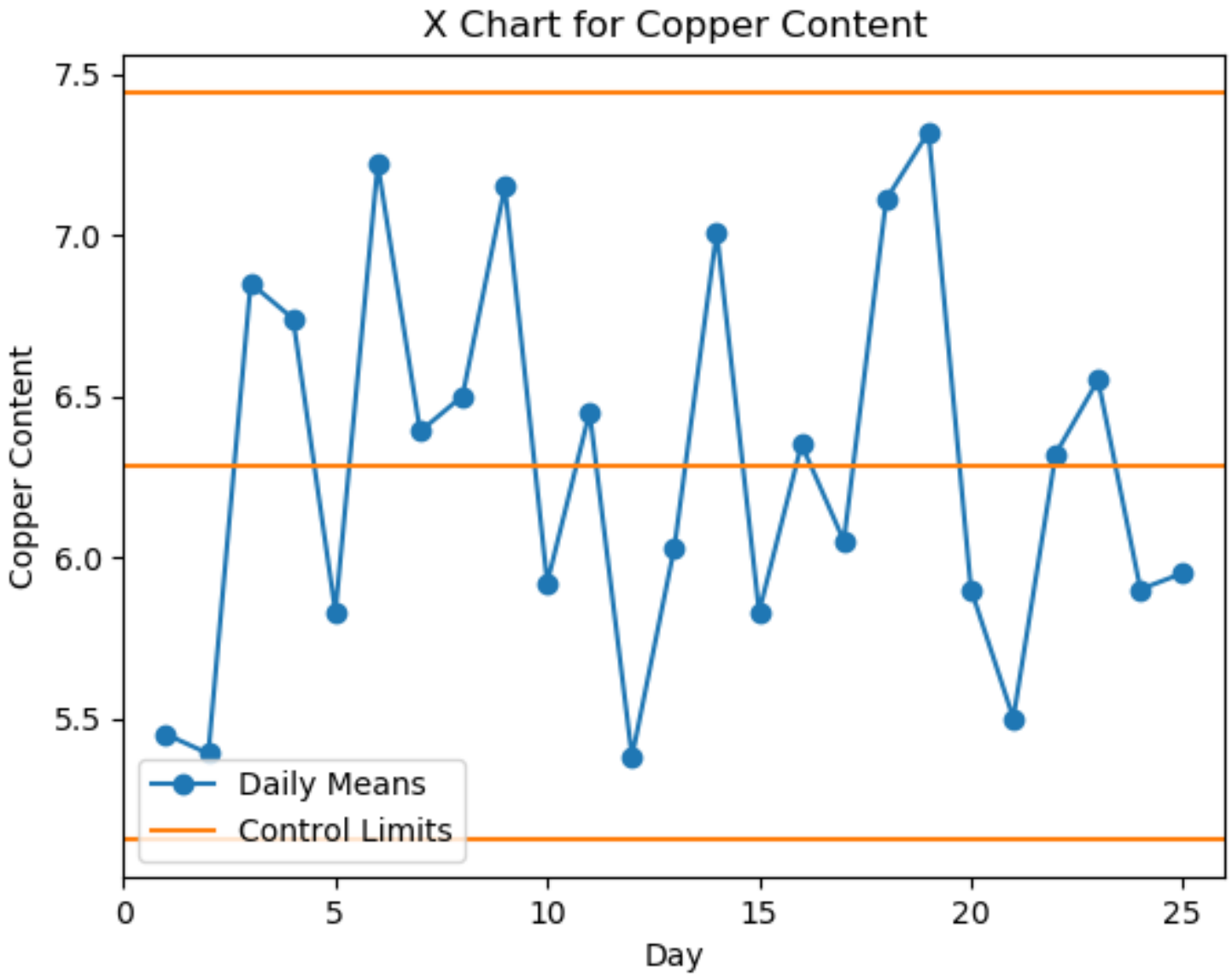
From appendix XI, we have

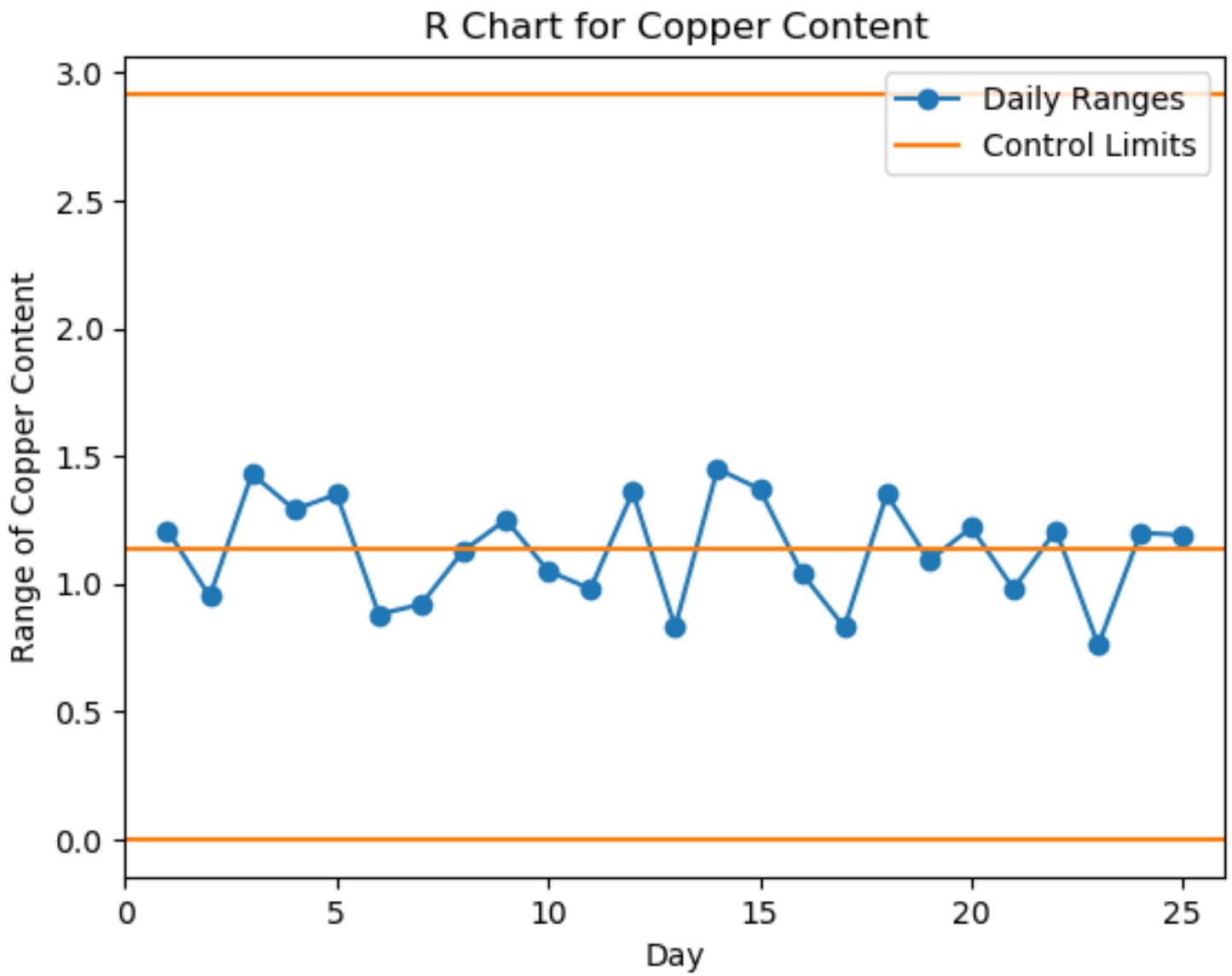
In [ ]:
```
A2 = 1.023
d2 = 1.693
D3 = 0
D4 = 2.575
```

In [ ]:
```
x_bar_bar = copper_data[:, 0].mean()
r_bar = copper_data[:, 1].mean()
x_bar_lcl = x_bar_bar - A2 * r_bar
x_bar_ucl = x_bar_bar + A2 * r_bar
r_lcl = D3 * r_bar
r_ucl = D4 * r_bar

plt.plot(
    np.array(list(range(1, num_measurements + 1))),
    copper_data[:,0],
    marker='o',
    label="Daily Means"
)
plt.plot(
    np.array([
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1
    ]),
    np.array([
        x_bar_bar,
```

```
        x_bar_bar,
        np.nan,
        x_bar_lcl,
        x_bar_lcl,
        np.nan,
        x_bar_ucl,
        x_bar_ucl
    ]),
    label="Control Limits"
)
plt.xlim([0, num_measurements+1])
plt.title("X Chart for Copper Content")
plt.xlabel("Day")
plt.ylabel("Copper Content")
plt.legend(loc="best")
plt.show()

plt.plot(np.array(list(range(1, num_measurements + 1))), copper_data[:,1], marker='o',
plt.plot(
    np.array([
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1
    ]),
    np.array([r_bar, r_bar, np.nan, r_lcl, r_lcl, np.nan, r_ucl, r_ucl]),
    label="Control Limits"
)
```

```
plt.xlim([0, num_measurements+1])
plt.title("R Chart for Copper Content")
plt.xlabel("Day")
plt.ylabel("Range of Copper Content")
plt.legend(loc="best")
plt.show()
```


X Chart for Copper Content
Copper Content
7.5
7.0
6.5
6.0
5.5
0
5
10
15
20
25
Day
Daily Means
Control Limits



From what the charts show, we can conclude that the process is in statistical control.

Part b

No changes needed to be made

Part c

In [ ]:
```
estimated_std = r_bar / d2
print("Estimated standard deviation:", estimated_std)
```

```
Estimated standard deviation: 0.669108092144123
```