

ENGS 93 HW 4

Cameron Wolfe 2/28/2024

```
In [ ]: import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
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

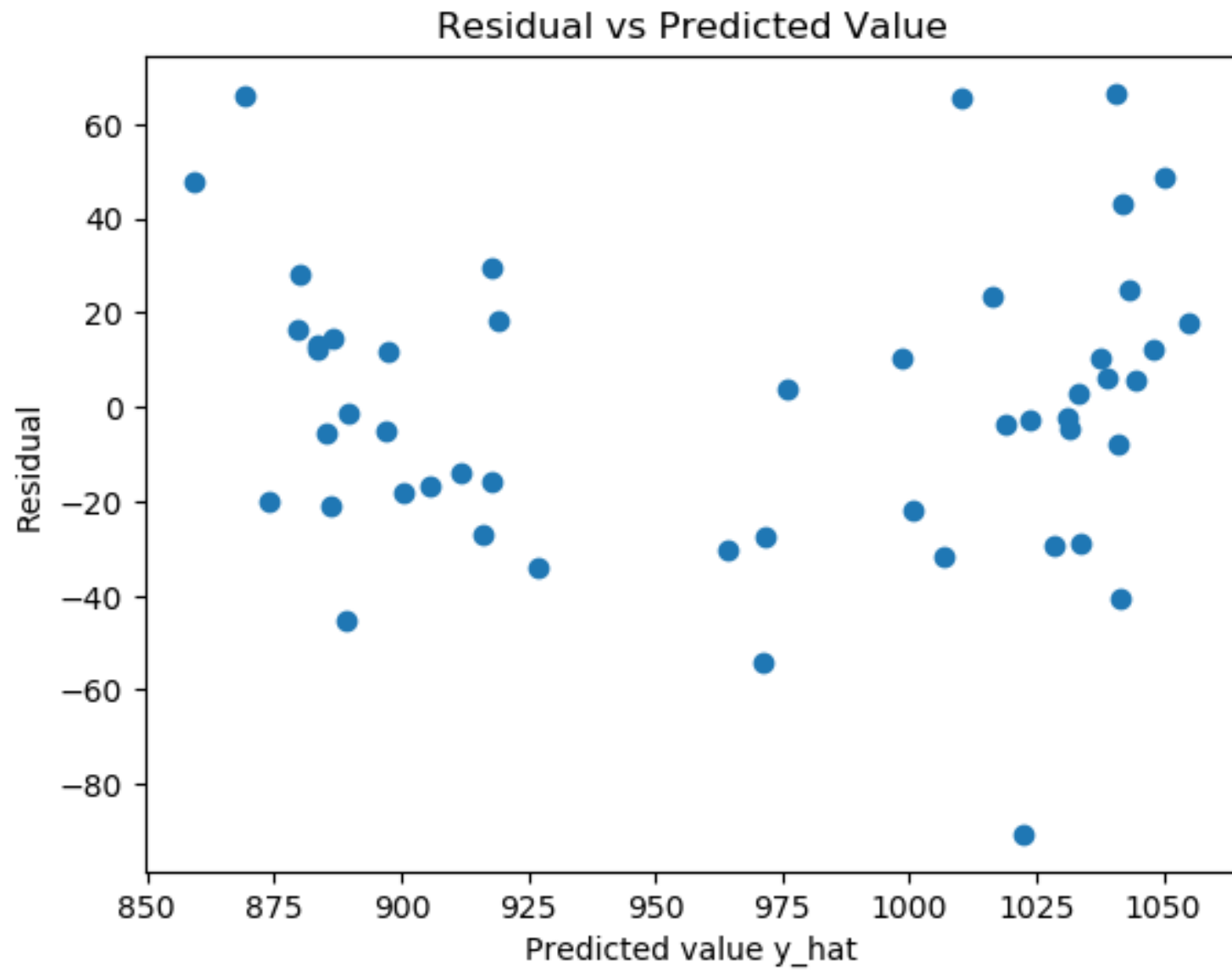
```
In [ ]: sat_data = np.genfromtxt("sat.csv", delimiter=",")[1:, 1:]
```

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

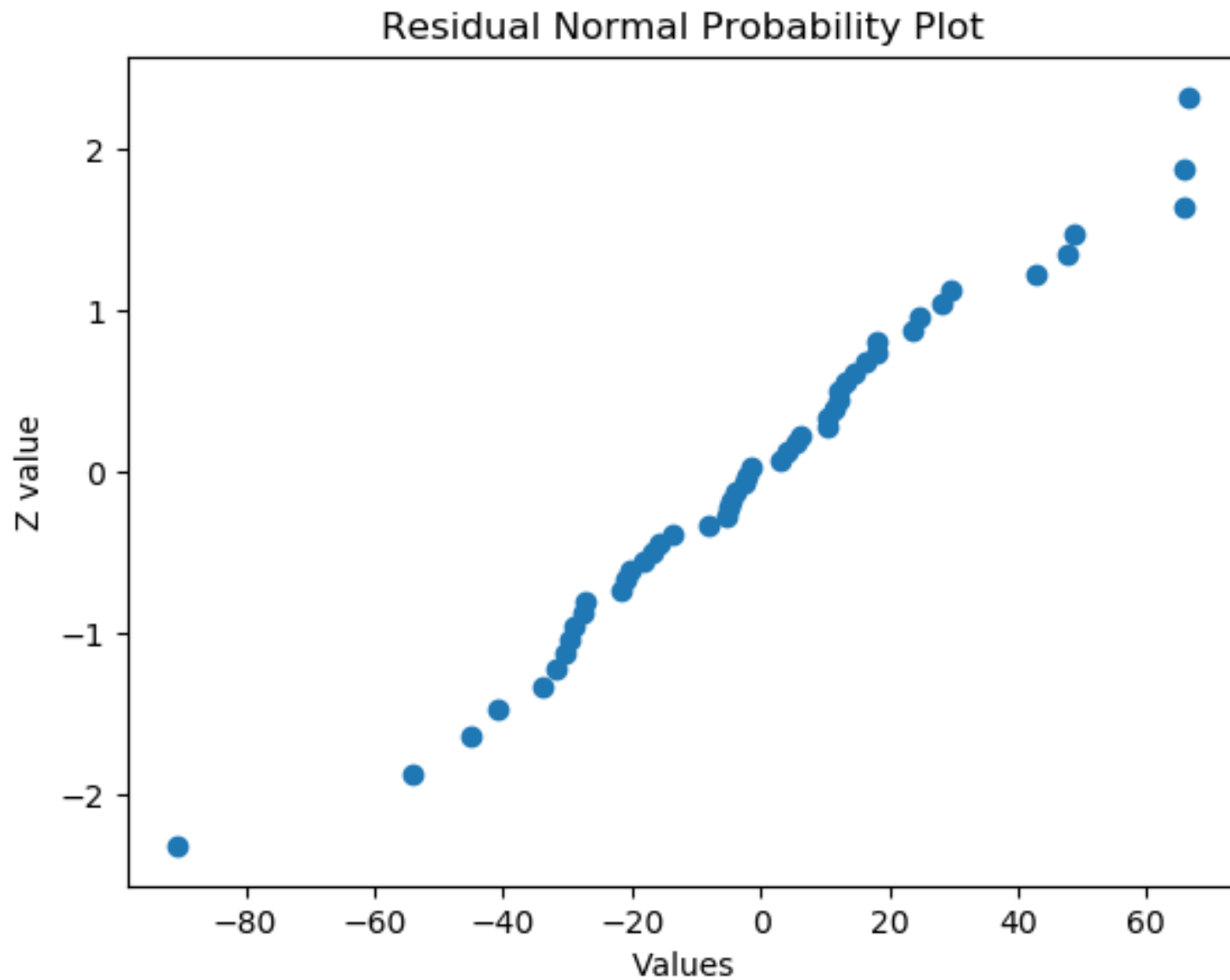
plt.scatter(y_hat, residual)
plt.title("Residual vs Predicted Value")
plt.xlabel("Predicted value y_hat")
plt.ylabel("Residual")
plt.show()
```



Variance appears to remain constant

Part b

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

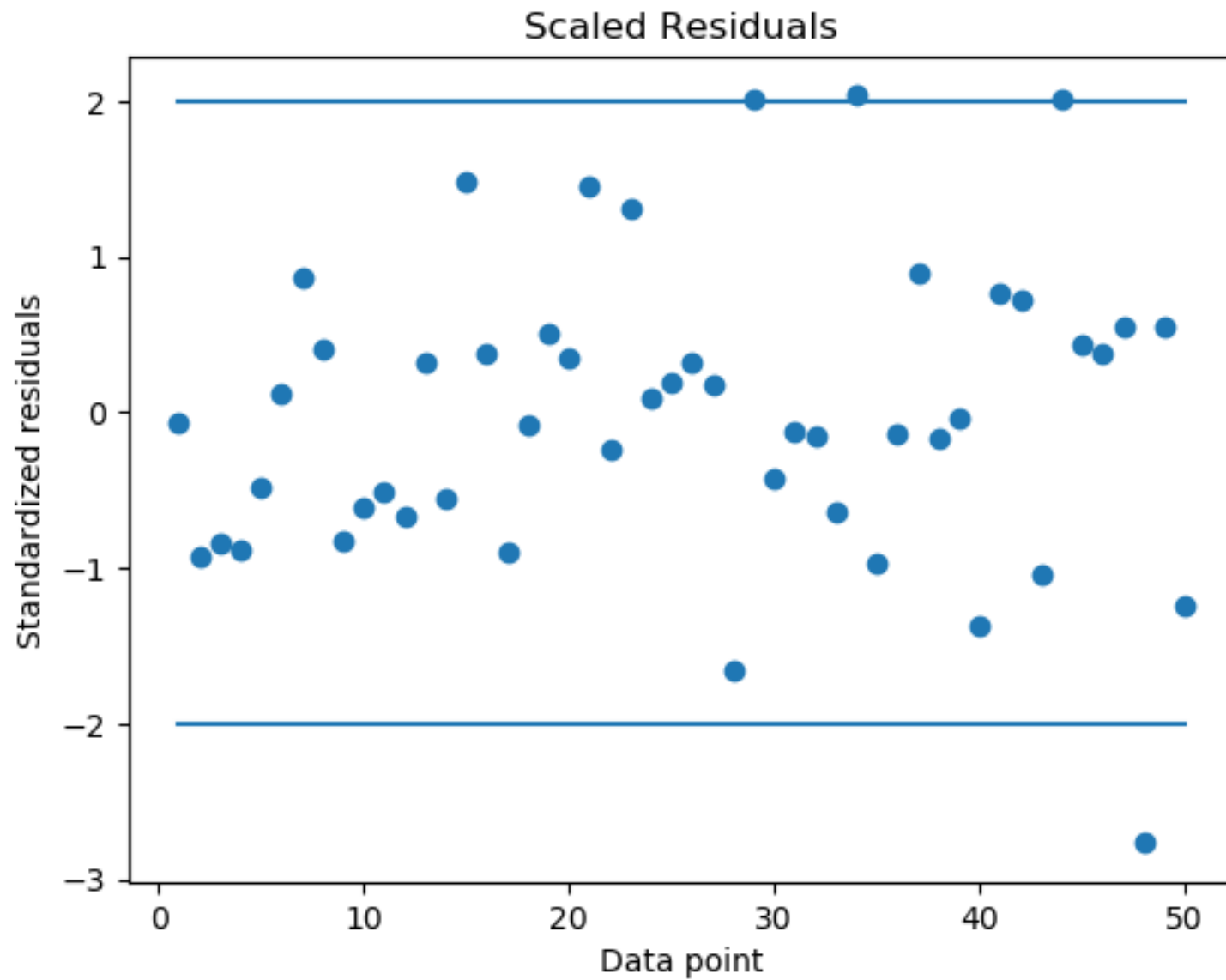


Residuals appear to be normally distributed

Part c

```
In [ ]: MSE = np.sum(residual ** 2) / (num_data_points - 5)
scaled_residual = residual / sqrt(MSE)
outliers = np.where(np.logical_or(scaled_residual > 2, scaled_residual < -2))
print("Outliers:", outliers)
plt.scatter(np.array(list(range(1, 51))), scaled_residual)
plt.title("Scaled Residuals")
plt.xlabel("Data point")
plt.ylabel("Standardized residuals")
plt.plot([1, 50, np.nan, 50, 1], [-2, -2, np.nan, 2, 2])
plt.show()
```

Outliers: (array([28, 33, 43, 47]),)



As we can see that there are 4 outliers with scaled residuals outside the range $[-2, 2]$. The outlier states are: Nevada, North Carolina, Texas, and 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, 1, n - p)

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

Test statistic: 0.29095563139930886

Critical value: 4.844335674943618

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:]
n = len(y)
p = 3
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 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 + -0.15212251914864716 * x_3

Part b

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

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

R^2: 0.9135769044606818

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)
SSE_reduced = np.sum(new_residual ** 2)
alpha = 0.05
F_0 = ((SSE_reduced - SSE) / r) / (SSE / (n - p - 1))
f_alpha = f.ppf(1-alpha, r, n - p - 1)

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

```

Test statistic: 28.160141604334104
Critical value: 4.451321772468133

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 = 4
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 [ ]: residual = y - np.matmul(x, beta)
SSE = np.sum(residual ** 2)
SST = np.sum(y ** 2) - y.mean() ** 2 * n
SSR = SST - SSE

F0 = (SSR / p) / (SSE / (n - p - 1))
p = f.sf(F0, p, n - p - 1)
print("P-value:", p)
```

P-value: 2.0815942472805593e-14

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 [ ]: 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)
SSE_reduced = np.sum(new_residual ** 2)
```

```

r = 2
f_val = (SSE_reduced - SSE) / r / (SSE / (n - p - 1))
p = f.sf(f_val, r, n-p-1)
print("P-value:", p)

```

P-value: 3.844408181787328e-16

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 - I)
print("Within group variance:", MSE)
```

Within group variance: 2.8579999999999974

Part b

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

F-statistic: 5.85206438068575

Part c

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

P-value: 0.00018131623842649586

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 [ ]: num_factors = 2
levels_A = fatigue_crack_data.shape[0]
levels_B = fatigue_crack_data.shape[1]
replicates = fatigue_crack_data.shape[2]

df = pd.DataFrame({
    'value': fatigue_crack_data.flatten(),
    'Frequency': np.repeat(np.arange(levels_A), levels_B*replicates),
    'Environment': np.tile(np.repeat(np.arange(levels_B), replicates), levels_A)
})

model = ols('value ~ C(Frequency) + C(Environment) + C(Frequency):C(Environment)', data=df)
anova_table = sm.stats.anova_lm(model, typ=2)

print(anova_table)

```

	sum_sq	df	F	PR(>F)
C(Frequency)	209.892822	2.0	522.398193	2.611529e-22
C(Environment)	64.252156	2.0	159.915950	1.075979e-15
C(Frequency):C(Environment)	101.965794	4.0	126.890349	4.324070e-17
Residual	5.424125	27.0	NaN	NaN

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

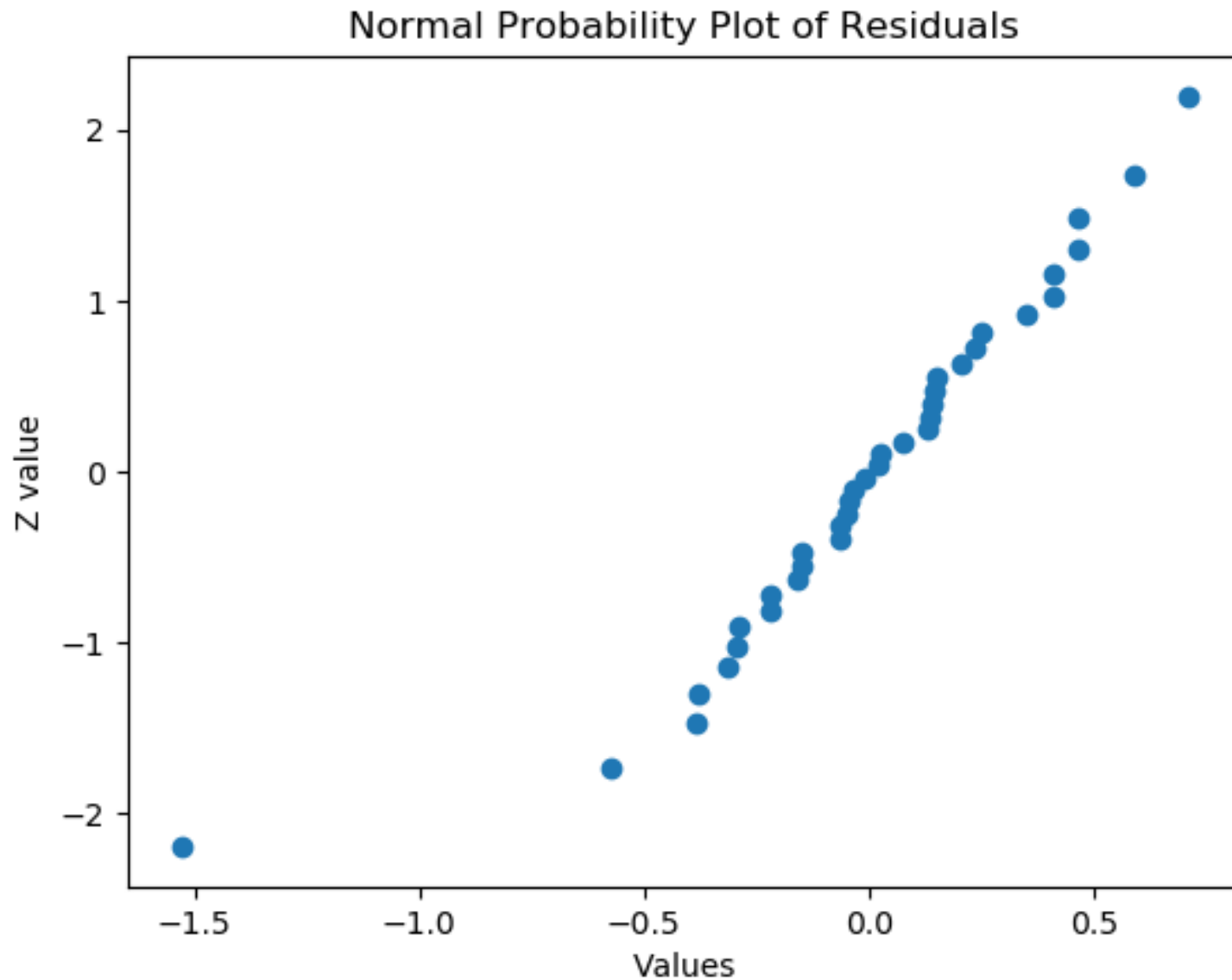
```
In [ ]: treatment_means = np.mean(fatigue_crack_data, 2)
row_means = np.mean(treatment_means, 1)
column_means = np.mean(treatment_means, 0)
grand_mean = np.mean(row_means)

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



Residuals appear to be normally distributed, with one outlier.

```

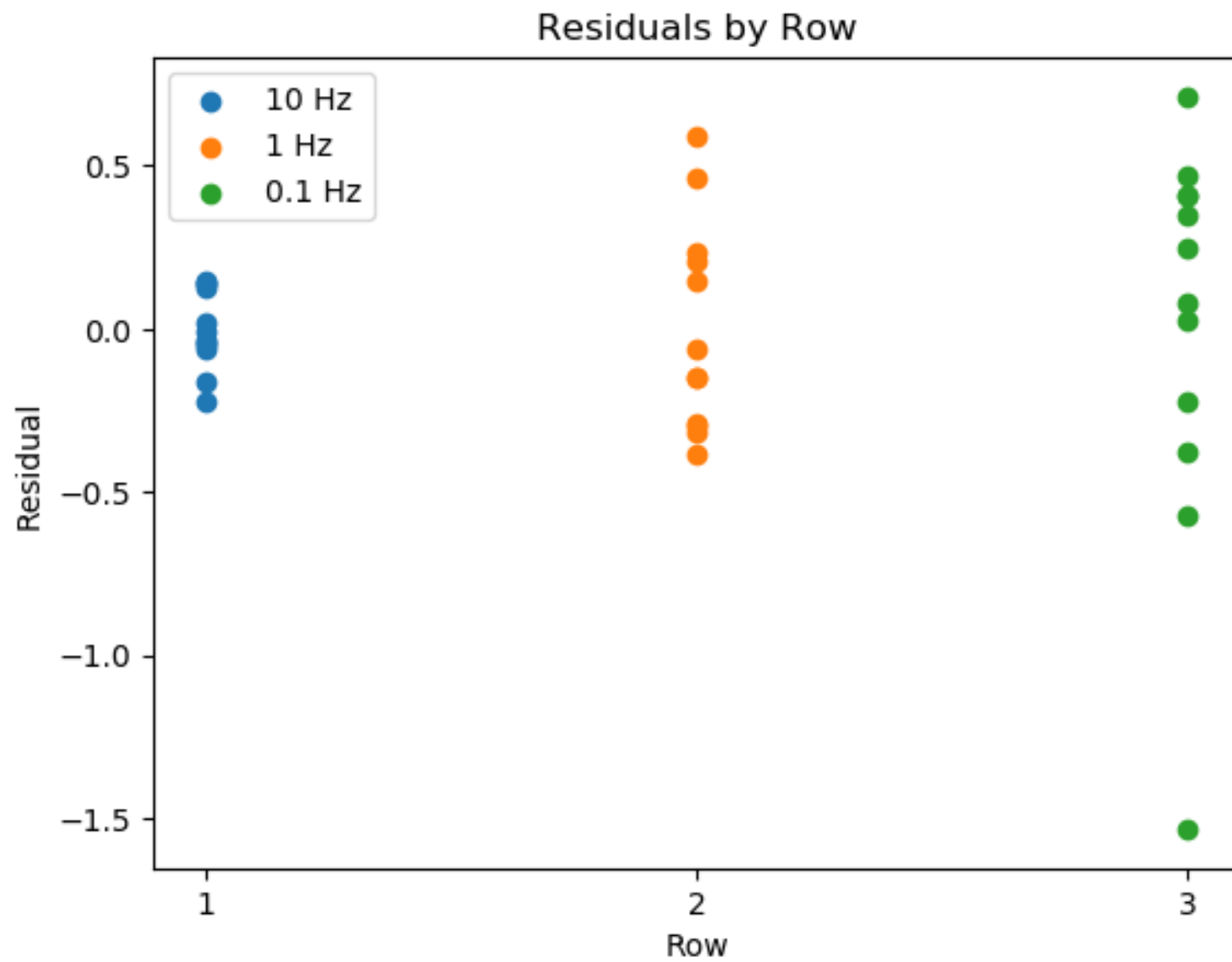
In [ ]: frequency_values = ["10 Hz", "1 Hz", "0.1 Hz"]
environment_types = ["Air", "Water", "Salt Water"]
residuals_by_row = []
residuals_by_column = []
residuals_by_treatment = []
for i in range(I):
    residuals_by_row.append(error[i, :, :].reshape((-1)))
    label_x = np.full((J*K), str(i+1), dtype=str)
    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()

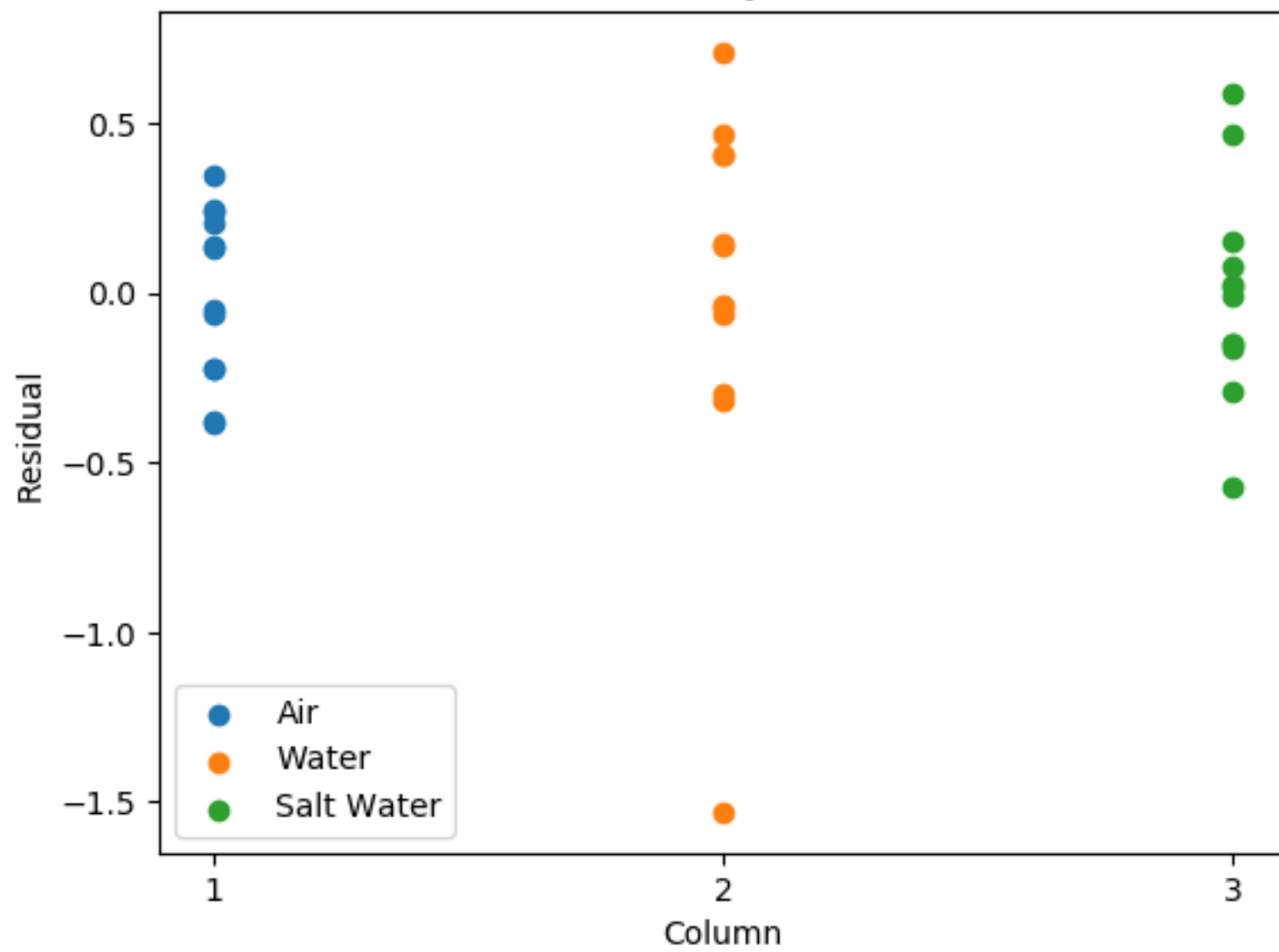
for i in range(I):
    for j in range(J):
        residuals_by_treatment.append(error[i,j,:].reshape((-1)))
        label_x = np.full((K), str(3 * i + j + 1), dtype=str)
        plt.scatter(label_x, residuals_by_treatment[-1], label=(frequency_values[i] +
plt.title("Residuals by Treatment")

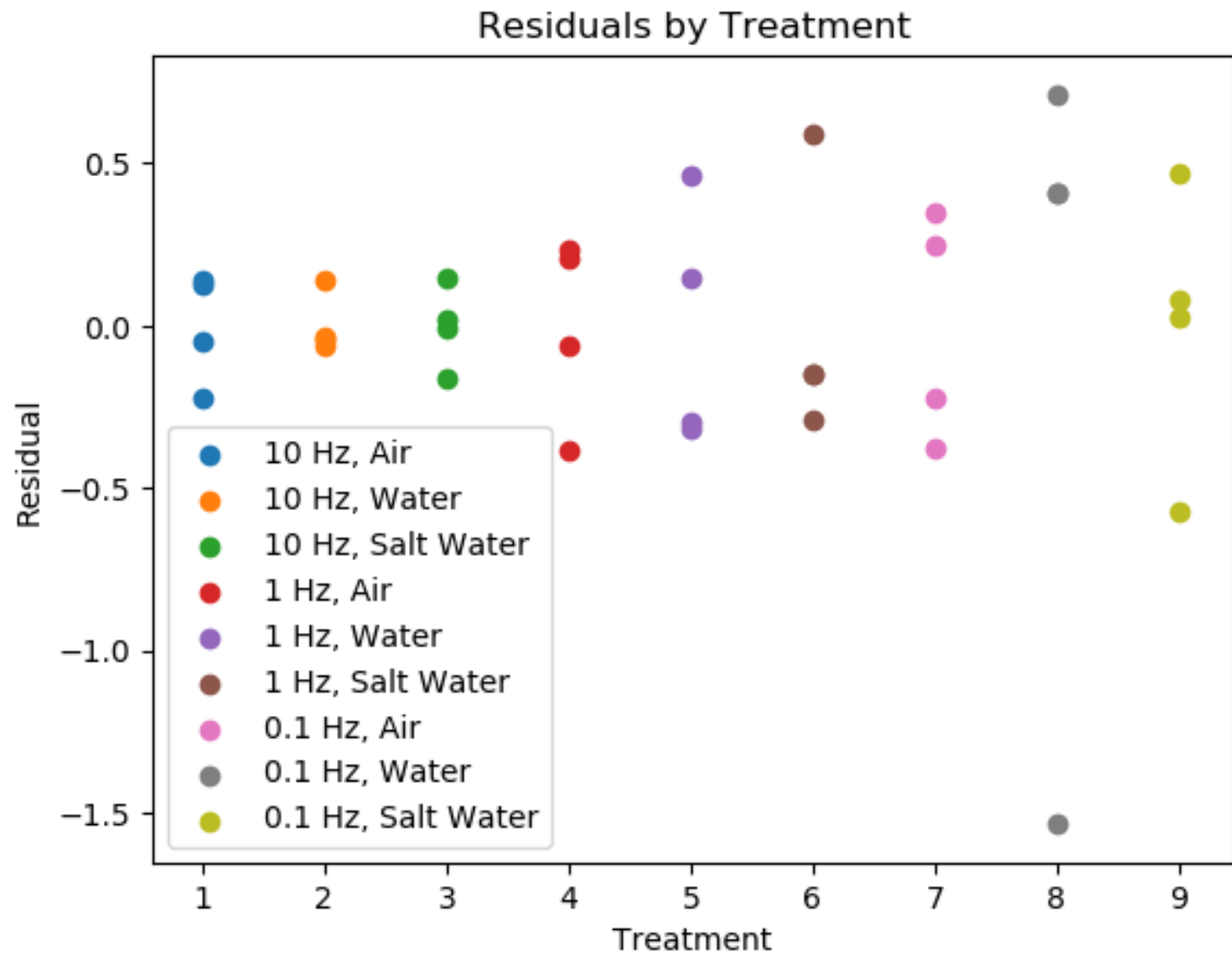
```

```
plt.xlabel("Treatment")  
plt.ylabel("Residual")  
plt.legend(loc="best")  
plt.show()
```



Residuals by Column





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)
df = pd.DataFrame({
    'value': fatigue_crack_data.flatten(),
    'Frequency': np.repeat(np.arange(levels_A), levels_B*replicates),
    'Environment': np.tile(np.repeat(np.arange(levels_B), replicates), levels_A)
})

model = ols('value ~ C(Frequency) + C(Environment) + C(Frequency):C(Environment)', data=df)
anova_table = sm.stats.anova_lm(model, typ=2)

print(anova_table)

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]

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]

```

	sum_sq	df	F	PR(>F)
C(Frequency)	7.570235	2.0	404.094970	7.573731e-21
C(Environment)	2.357628	2.0	125.848875	2.061148e-14
C(Frequency):C(Environment)	3.528400	4.0	94.172027	1.884673e-15
Residual	0.252906	27.0	NaN	NaN

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")

```

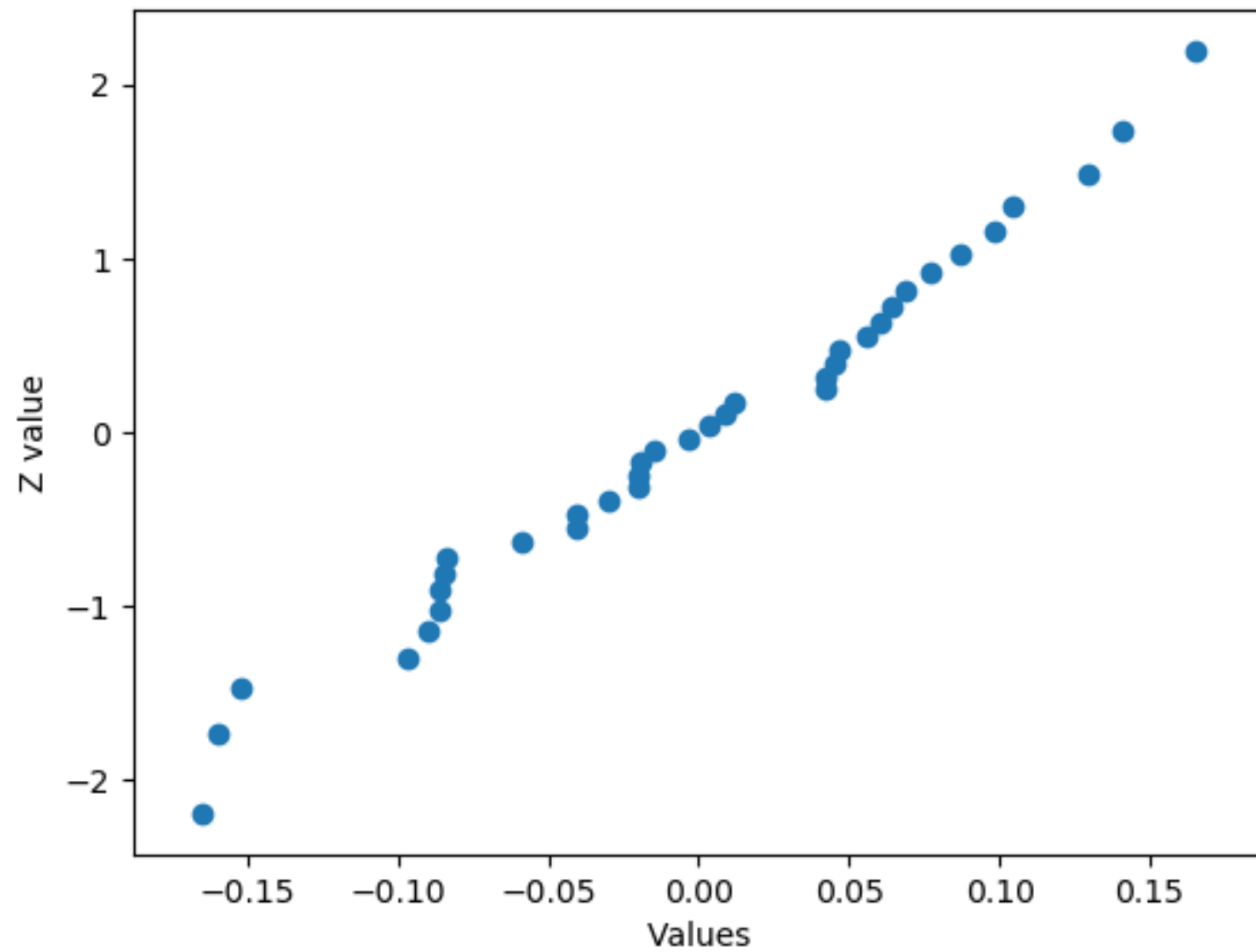
```

plt.xlabel("Column")
plt.ylabel("Residual")
plt.legend(loc="best")
plt.show()

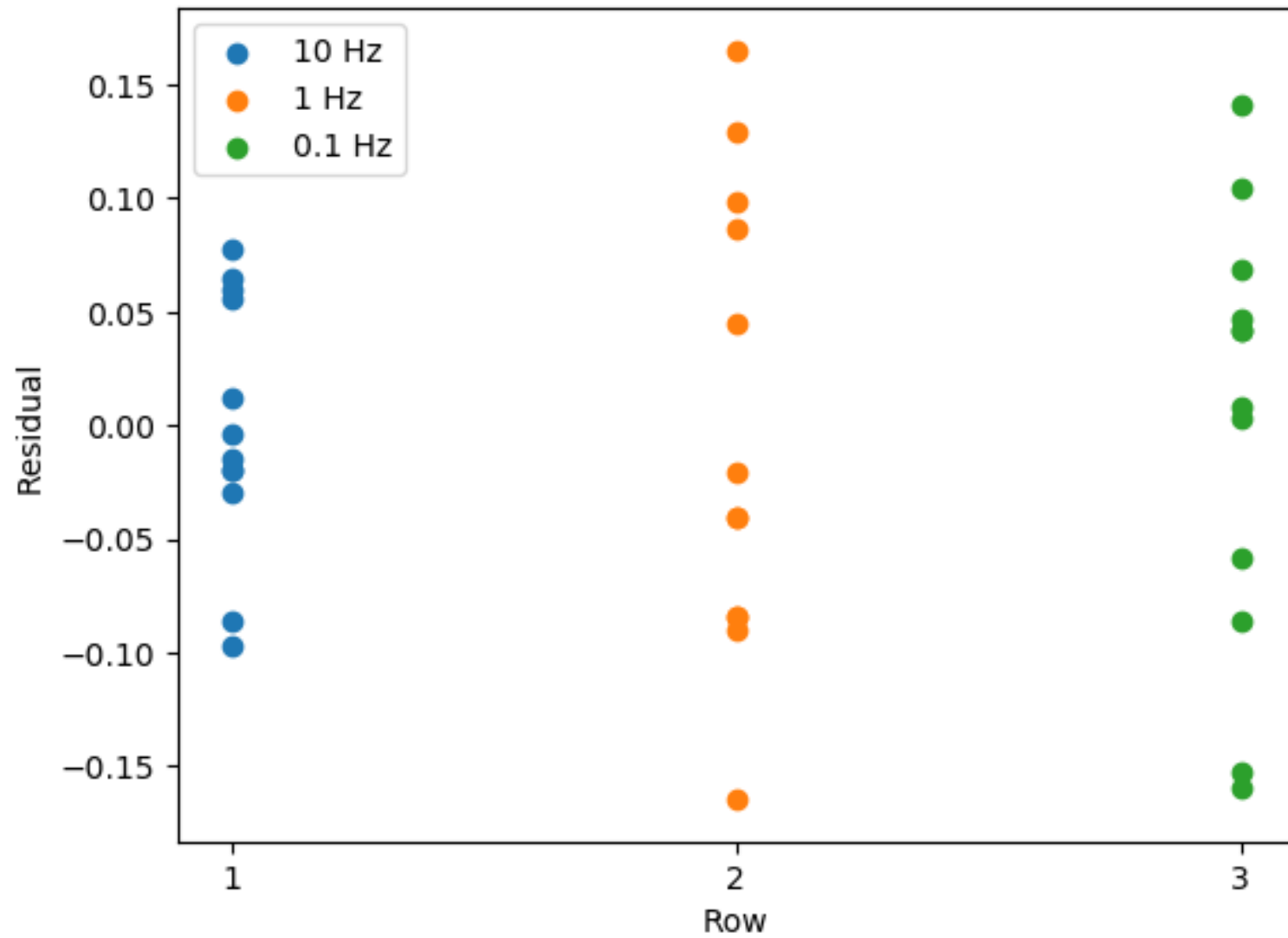
for i in range(I):
    for j in range(J):
        residuals_by_treatment.append(error[i,j,:].reshape((-1)))
        label_x = np.full((K), str(3 * i + j + 1), dtype=str)
        plt.scatter(
            label_x, residuals_by_treatment[-1],
            label=(frequency_values[i] + ", " + environment_types[j]))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

```

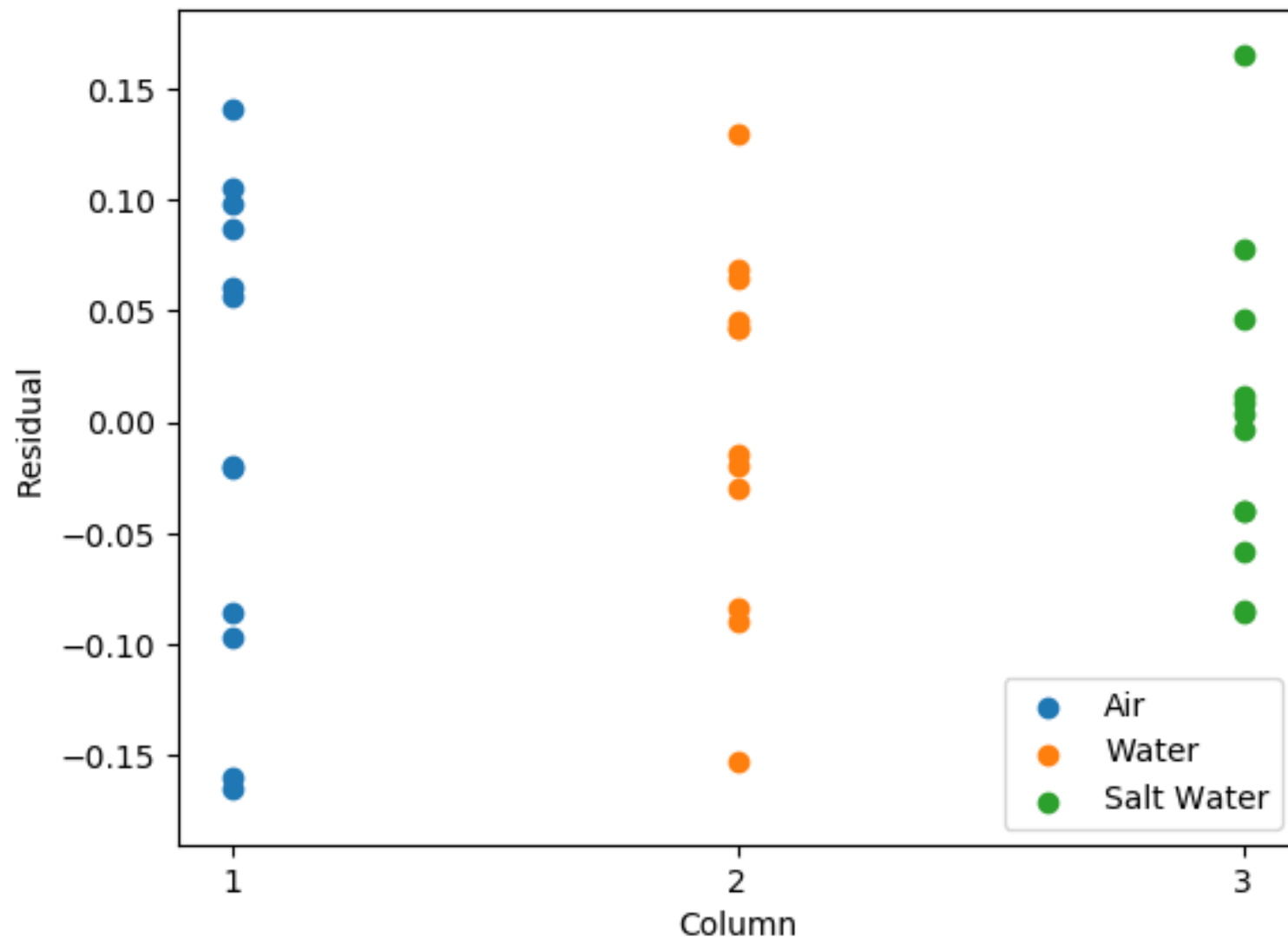

Normal Probability Plot of Residuals

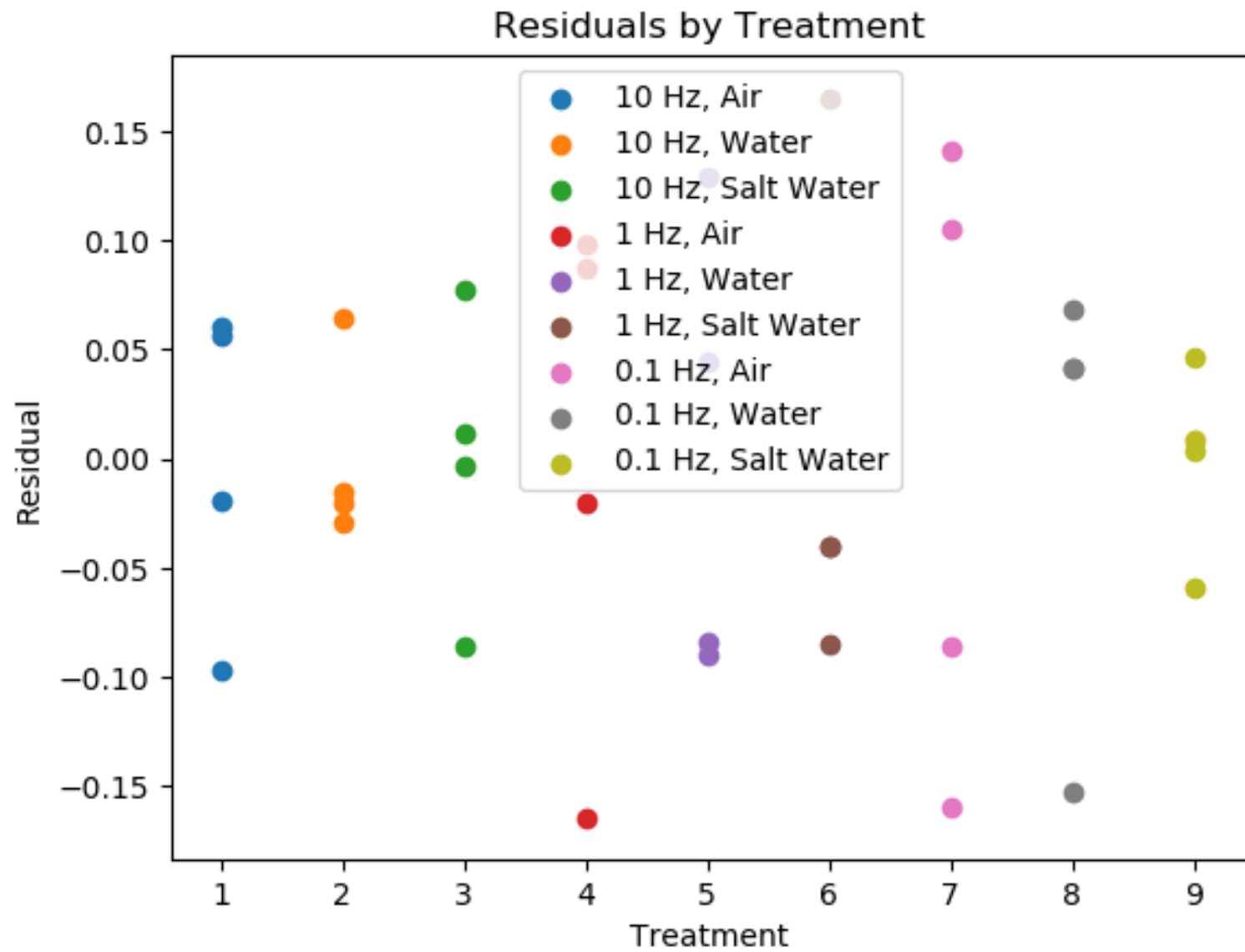


Residuals by Row



Residuals by Column





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

print(new_SSE, new_MSE)
p = f.sf(F, I - 1, N - I)
print("P-value:", p)
```

502 8.096774193548388

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()

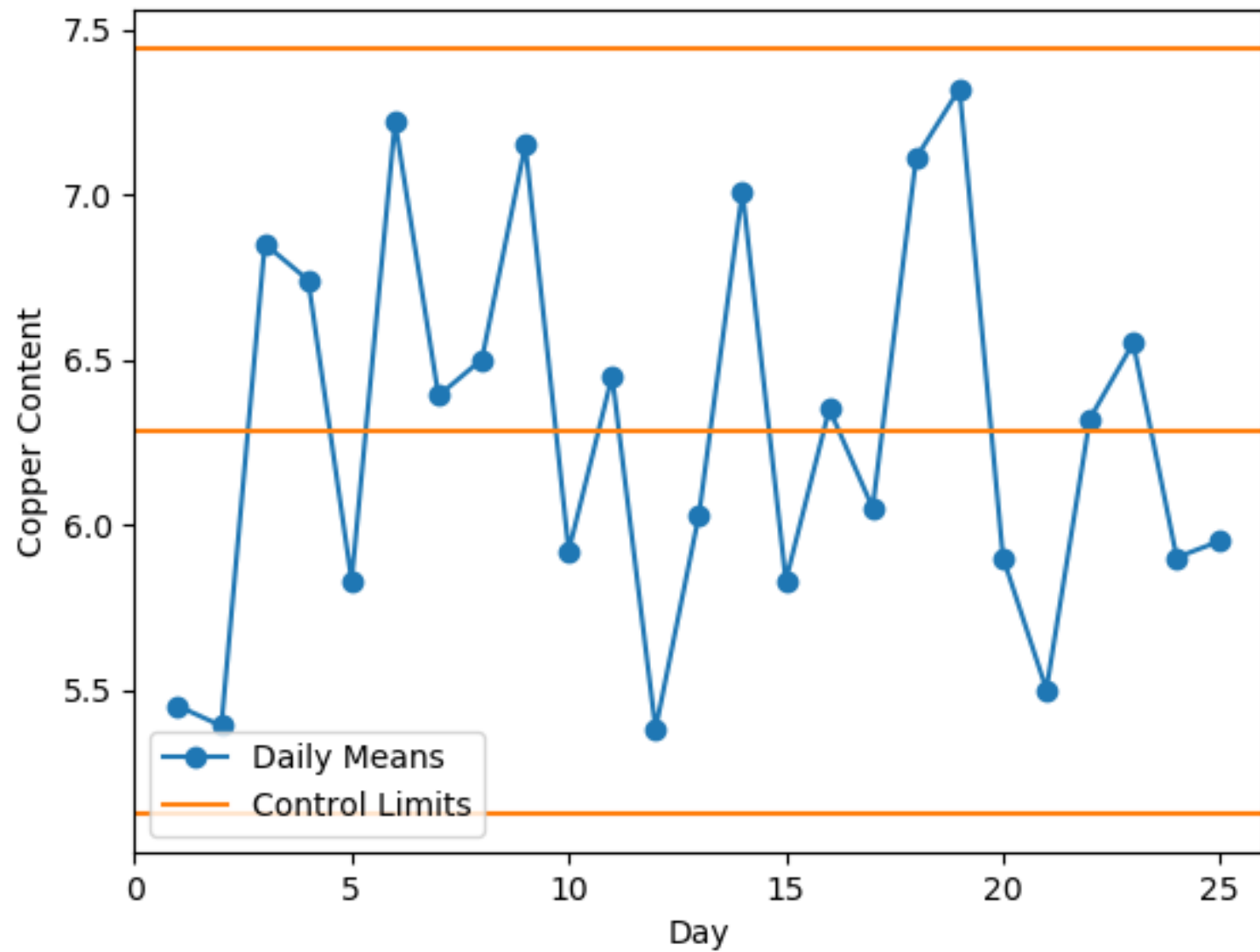
plt.plot(np.array(list(range(1, num_measurements + 1))), copper_data[:,1], marker='o',
plt.plot(
    np.array([
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1,
        np.nan,
        0,
        num_measurements+1
    ]),
    np.array([r_bar, r_bar, np.nan,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

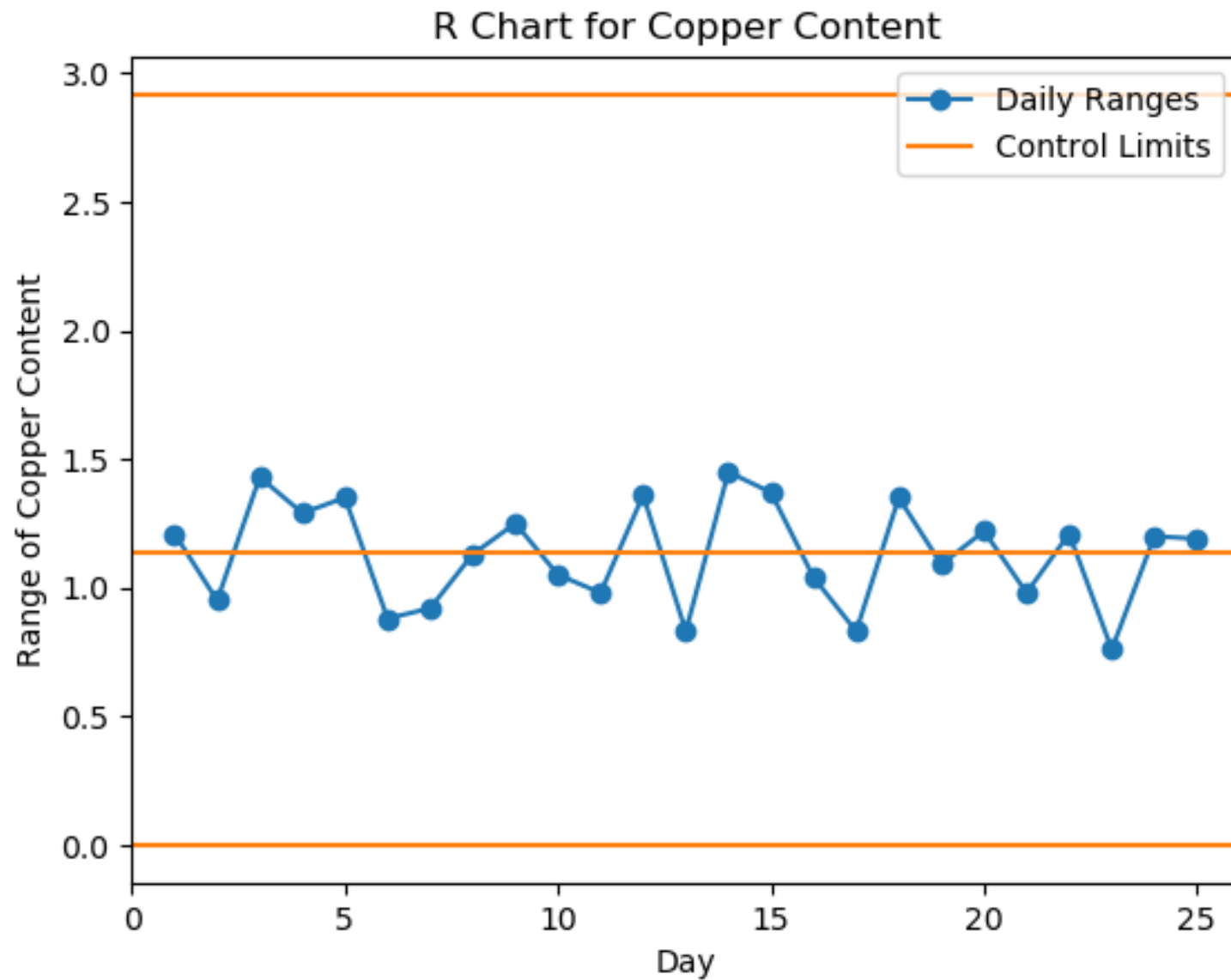
```



```
        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





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