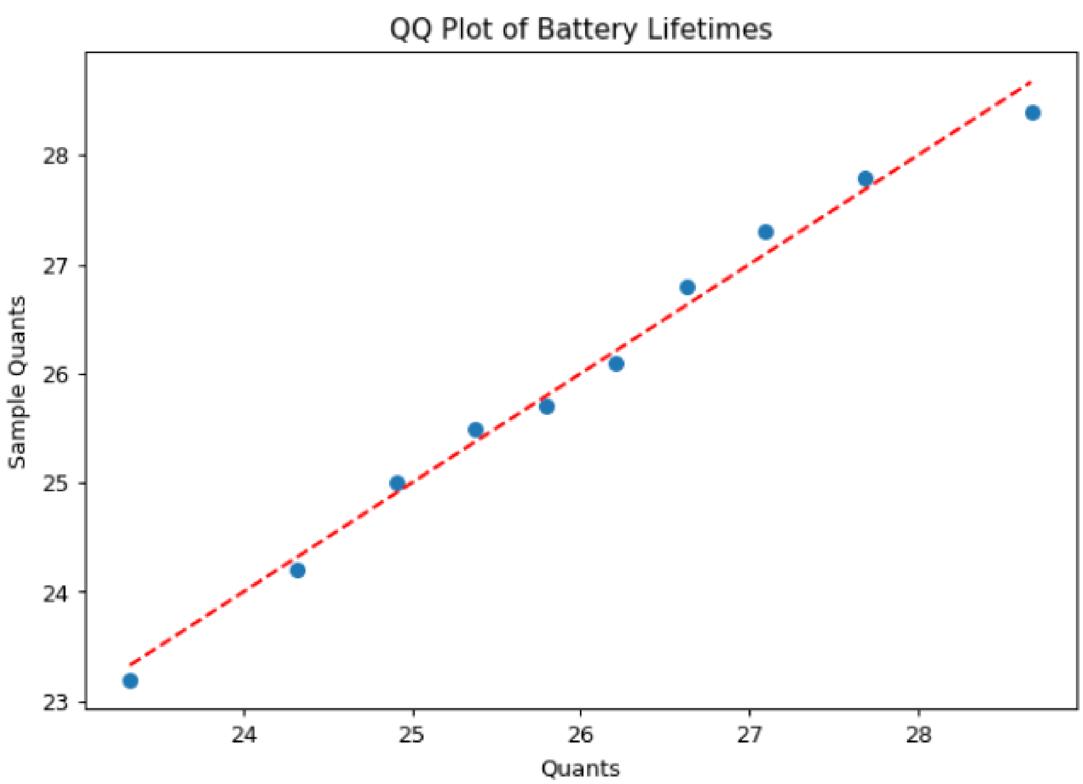
```
import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm # For lm() equivalent (see line 322)

import pandas as pd

import seaborn as sns

from statsmodels.stats.anova import anova_lm # For anova() equivalent (see line 330)

from statsmodels.formula.api import ols # For lm() equivalent (see line 339)

import scipy.stats as stats

from scipy.stats import hypergeom

from scipy.stats import binom

from scipy.stats import poisson

from scipy.stats import norm

from scipy.stats import chi2 # For pchisq() equivalent (see line 111)

from scipy.stats import f # For pf() equivalent (see line 129)

from scipy.stats import t # For pt() equivalent (see line 147)

from scipy.stats import ttest_1samp # For one-sample t.test() equivalent (see line 157)

from scipy.stats import ttest_ind # For two-sample t.test() equivalent (see line 189)

from scipy.stats import ttest_rel # For paired two-sample t.test() equivalent (see line 199)

from scipy.stats import binomtest # For one-sample prop.test() equivalent (see line 265)

from statsmodels.stats.proportion import proportions_ztest # For two-sample prop.test()
equivalent (see line 274)


#Problem 1.A

Lifetimes = [25.5, 26.1, 26.8, 23.2, 24.2, 28.4, 25.0, 27.8, 27.3, 25.7]

t_stat, p_value = ttest_1samp(Lifetimes, popmean=25)

print(p_value/2)
```

```
print (t_stat)

#P/2 is <.05 and T>0 therefore mean battery life is above 25Hrs


#Problem 1.B

n = len(Lifetimes)

mean = np.mean(Lifetimes)

std = np.std(Lifetimes, ddof=1)

confidence = 0.90

alpha = 1 - confidence

df = n - 1

t_crit = t.ppf(1 - alpha/2, df)

margin_error = t_crit * (std / np.sqrt(n))

lowerInterval = mean - margin_error

upperInterval = mean + margin_error


print(f"90% Confidence Interval: ({lowerInterval:.2f}, {upperInterval:.2f})")

#This means that there is a 90% chance that the mean of the total population of batteries
between 25.06 and 26.94


#Problem 1.C

plt.close()

lifetimesSorted = np.sort(Lifetimes)

p = [(i - .5) / n for i in range(1, n+1)]

quants = norm.ppf(p, loc=mean, scale=std)

plt.scatter(quants, lifetimesSorted)

plt.plot(quants, quants, color='red', linestyle='--')  # reference line
```

```
plt.title("QQ Plot of Battery Lifetimes")

plt.xlabel("Quants")

plt.ylabel("Sample Quants")

plt.show()

#Values are close to a straight line so they appear to be almost normally distributed.

#Therefore, t_test and confidence intervals can be used assuming they are normal.
```

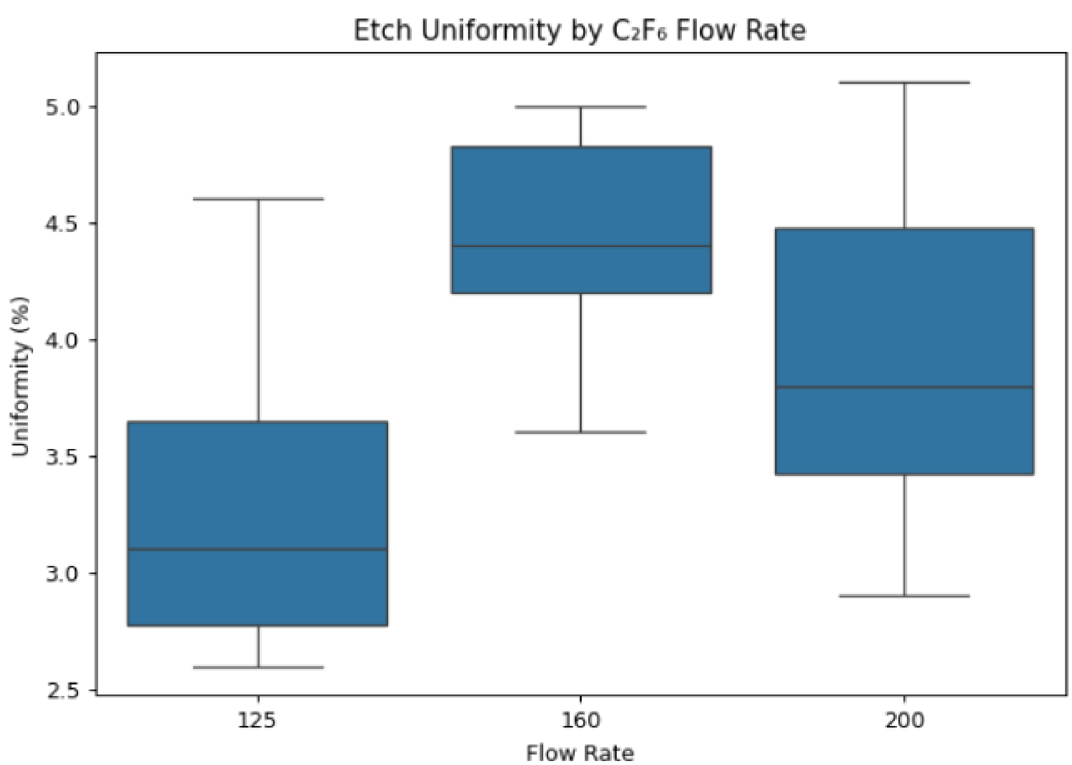



```
#Problem 2.A

n = 500

x = 65

fraction = x / n

alpha = .1
```

```
p0 = .08

std0 = np.sqrt(p0 * (1-p0)/n)

z = (fraction - p0) / std0

p = 2 * (1 - norm.cdf(abs(z)))

print(f"P:{p:.2f} < Alpha:{alpha:.2f}?")

#Null Hypothesis Rejected. Strong evidence that is it much higher than .08


#Problem 2.B

z = norm.ppf(.95)

upperInterval = fraction + z * std0

print(upperInterval)

#around .15


#Problem 3

alpha = .01

Micrometer= [0.150,0.151,0.151,0.152,0.151,0.150,0.151,0.153,0.152,0.151,0.151,0.151]

Vernier= [0.151,0.150,0.151,0.150,0.151,0.151,0.153,0.155,0.154,0.151,0.150,0.152]

t_stat, p_stat = ttest_ind(Micrometer, Vernier)

print(f"P:{p_stat:.2f} < Alpha:{alpha:.2f}?")

#p is .44 so we fail to reject Null Hype. No significant mean difference.


#Problem 4.A

Flow = [125,125,125,125,125,125,160,160,160,160,160,160,200,200,200,200,200,200]

Uniformity = [2.7,2.6,4.6,3.2,3,3.8,4.6,4.9,5,4.2,3.6,4.2,4.6,2.9,3.4,3.5,4.1,5.1]

df = pd.DataFrame({'Flow': Flow, 'Uniformity': Uniformity})
```

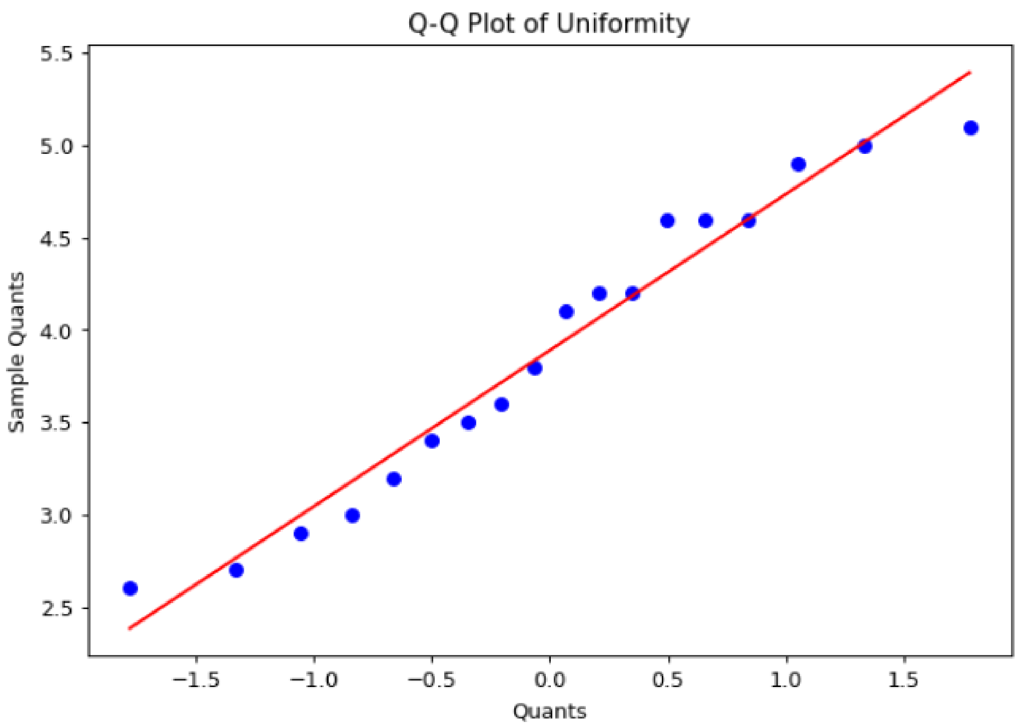
```
# Convert Flow to Category and Create Dummies

df = pd.get_dummies(df, columns=['Flow'], drop_first=True) # Drops one category to avoid multicollinearity (similar to R’s default behavior with factors in linear regression).

print(df.head())


# Convert Dummy Variables to 1 and 0

df['Flow_160'] = df['Flow_160'].astype(int)

df['Flow_200'] = df['Flow_200'].astype(int)



# Define the response variable and predictors

X = df.drop(columns='Uniformity')  # Predictor variables

y = df['Uniformity']             # Response variable


# Add a constant to the predictors (intercept term)

X = sm.add_constant(X)


# Fit the linear regression model

model = sm.OLS(y, X)

results = model.fit()


# Print the summary (similar to R's summary(lm(...)))

print(results.summary())

#F-Stat is .0534. So it's above .05 so we do not reject h0.So there is

#Not enough to conclude C2F6 Flow Rate significantly affects etch uniformity.
```

```
#Problem 4.B

plt.close()

df = pd.DataFrame({'Flow': Flow, 'Uniformity': Uniformity})

plt.figure(figsize=(8, 6))

sns.boxplot(x='Flow', y='Uniformity', data=df)

plt.title("Etch Uniformity by C2F6 Flow Rate")

plt.xlabel("Flow Rate")

plt.ylabel("Uniformity (%)")

plt.show()

#125. it has the lowest values of the 3.
```



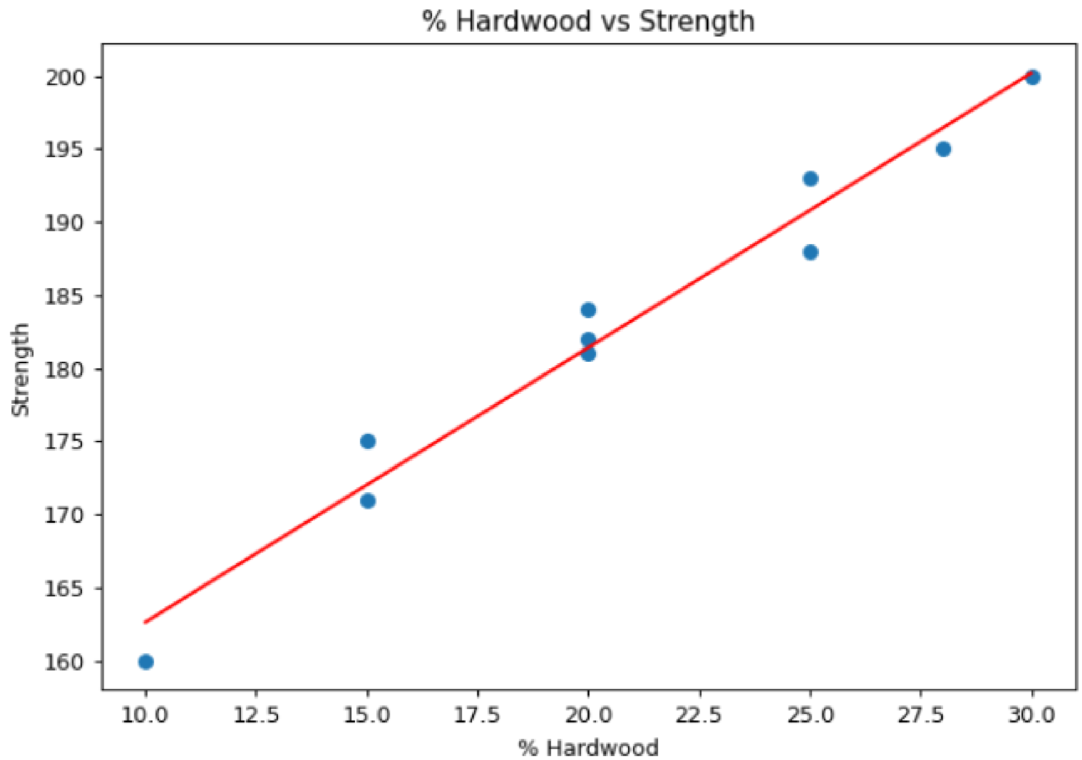

```
#Problem 4.C

plt.close()
```

```
stats.probplot(Uniformity, dist="norm", plot=plt)

plt.title("Q-Q Plot of Uniformity")

plt.xlabel("Quants")

plt.ylabel("Sample Quants")

plt.show()

#Yes, looks close to normally distributed
```




```
#Problem 5.A

Strength = [160,171,175,182,184,181,188,193,195,200]

PctHardwood = [10,15,15,20,20,20,25,25,28,30]

df = pd.DataFrame({'PctHardwood': PctHardwood, 'Strength': Strength})

df = pd.get_dummies(df, columns=['PctHardwood'], drop_first=True)

model = ols('Strength ~ PctHardwood', data=df).fit()

model.params
```

```
# y = 1.8786 * x + 143.82


#Problem 5.B

model.summary()

#p is close to 0, so H0 is rejected and the Strength is related to amount

#of Hardwood


#Problem 5.C

m = 1.8786

b = 143.82

line = [m * x + b for x in PctHardwood]

plt.close()

plt.scatter(PctHardwood, Strength, marker='o')

plt.plot(PctHardwood, line, color='red')

plt.title("% Hardwood vs Strength")

plt.xlabel("% Hardwood")

plt.ylabel("Strength")

plt.show()
```



#Problem 5.D

#As the % of Hardwood goes up, the strength of it goes up in a linear way.

#1% increase in HArdwood gives 1.88 Strength units

#p value being very low confirms this relationship is significant