Practical Part 2

Q.No.1 Write a Python program to perform a Z-test for comparing a sample mean to a known population mean and interpret the results.
Ans-

```
from scipy.stats import norm
import numpy as np

sample = np.random.normal(100, 15, 50)
sample_mean = np.mean(sample)
population_mean = 105
population_std = 15

z = (sample_mean - population_mean) / (population_std / np.sqrt(len(sample)))
p_value = 2 * (1 - norm.cdf(abs(z)))

print("Z-score:", z)
print("P-value:", p_value)
```

```
Z-score: -3.5202536326212877
P-value: 0.00043113431170804795
```

Q.No.2 Simulate random data to perform hypothesis testing and calculate the corresponding P-value using Python.
Ans-

```
[2] sample = np.random.normal(70, 10, 40)
    mean_sample = np.mean(sample)
    z = (mean_sample - 75) / (10 / np.sqrt(40))
    p_value = 2 * (1 - norm.cdf(abs(z)))

    print("Z-test P-value:", p_value)
```

```
Z-test P-value: 0.11212386153482767
```

Q.No.3 Implement a one-sample Z-test using Python to compare the sample mean with the population mean.
Ans-

```python
def one_sample_z_test(sample, population_mean, population_std):
    sample_mean = np.mean(sample)
    z = (sample_mean - population_mean) / (population_std / np.sqrt(len(sample)))
    p_value = 2 * (1 - norm.cdf(abs(z)))
    return z, p_value

sample = np.random.normal(55, 5, 30)
z, p = one_sample_z_test(sample, 50, 5)
print("Z:", z, "P-value:", p)
```

```
Z: 5.747415767513839 P-value: 9.061771555352038e-09
```

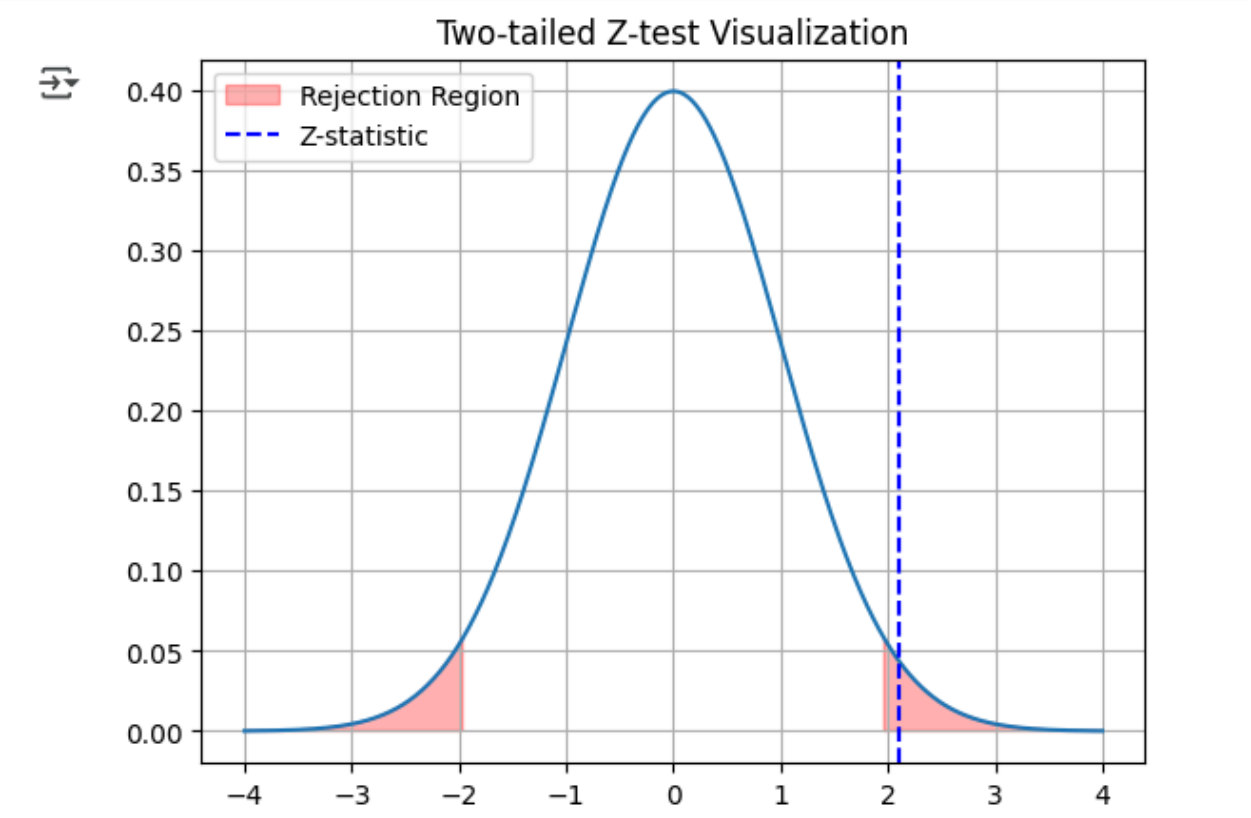
Q.No.4 Perform a two-tailed Z-test using Python and visualize the decision region on a plot.

Ans-

```python
import matplotlib.pyplot as plt

z_stat = 2.1  # example
x = np.linspace(-4, 4, 1000)
y = norm.pdf(x)

plt.plot(x, y)
plt.fill_between(x, y, where=(x < -1.96) | (x > 1.96), color='red', alpha=0.3, label='Rejection Region')
plt.axvline(z_stat, color='blue', linestyle='--', label='Z-statistic')
plt.title("Two-tailed Z-test Visualization")
plt.legend()
plt.grid(True)
plt.show()
```



Q.No.5 Create a Python function that calculates and visualizes Type 1 and Type 2 errors during hypothesis testing.

Ans-

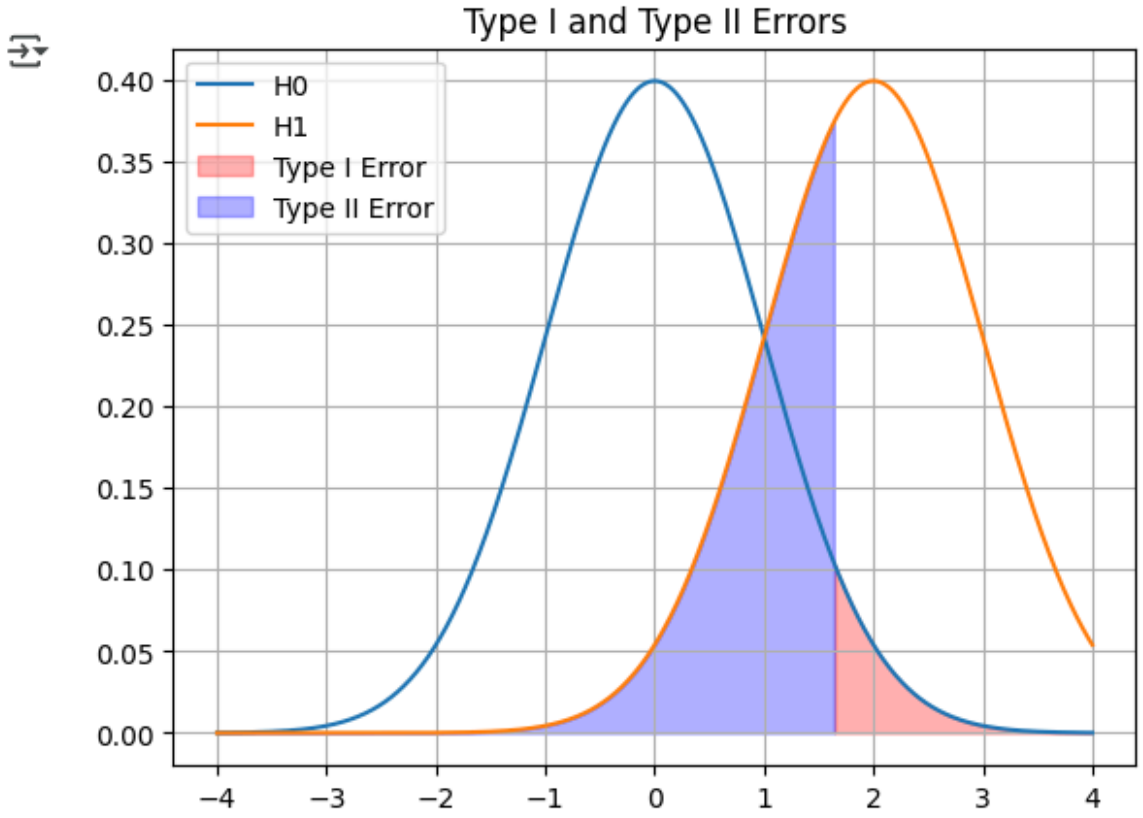
```
def visualize_errors():
    x = np.linspace(-4, 4, 1000)
    alpha = 0.05
    beta = 0.2
    critical = norm.ppf(1 - alpha)

    plt.plot(x, norm.pdf(x), label='H0')
    plt.plot(x, norm.pdf(x - 2), label='H1')  # shifted mean

    plt.fill_between(x, norm.pdf(x), where=(x > critical), color='red', alpha=0.3, label='Type I Error')
    plt.fill_between(x, norm.pdf(x - 2), where=(x < critical), color='blue', alpha=0.3, label='Type II Error')

    plt.legend()
    plt.title("Type I and Type II Errors")
    plt.grid(True)
    plt.show()

visualize_errors()
```



Q.No.6 Write a Python program to perform an independent T-test and interpret the results.
Ans-

```
[6]  from scipy.stats import ttest_ind

     group1 = np.random.normal(50, 10, 30)
     group2 = np.random.normal(55, 12, 30)

     t_stat, p_val = ttest_ind(group1, group2)
     print("T-statistic:", t_stat)
     print("P-value:", p_val)
```

```
T-statistic: -3.604127046264657
P-value: 0.0006516075046801693
```

Q.No.7 Perform a paired sample T-test using Python and visualize the comparison results.
Ans-

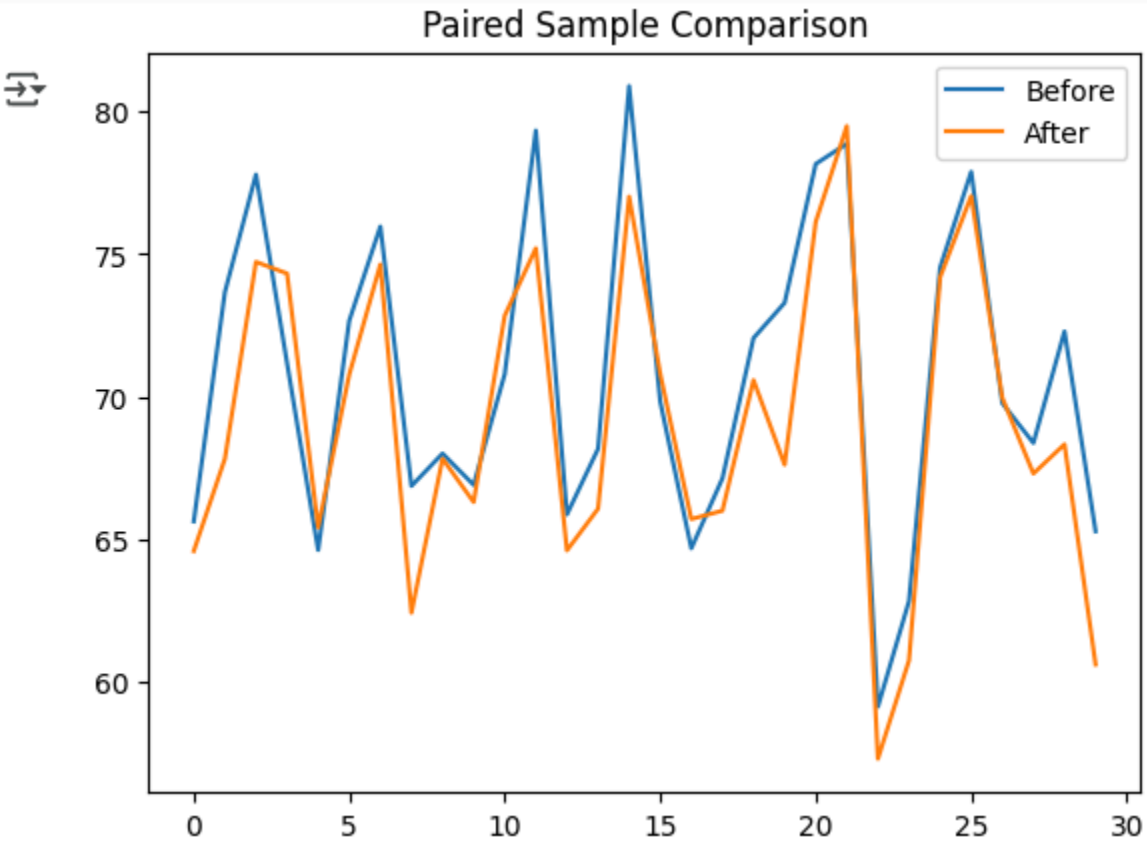
[7]
```
from scipy.stats import ttest_rel

before = np.random.normal(70, 5, 30)
after = before + np.random.normal(-2, 2, 30)

t_stat, p_val = ttest_rel(before, after)
print("Paired T-test T-statistic:", t_stat)
print("P-value:", p_val)

plt.plot(before, label='Before')
plt.plot(after, label='After')
plt.title("Paired Sample Comparison")
plt.legend()
plt.show()
```

```
Paired T-test T-statistic: 3.8116180569845586
P-value: 0.0006655708830438948
```



Q.No.8 Simulate data and perform both Z-test and T-test, then compare the results using Python.
Ans-

```
[8]  from scipy.stats import ttest_1samp

     sample = np.random.normal(100, 15, 10)
     mean = np.mean(sample)

     z_stat = (mean - 105) / (15 / np.sqrt(len(sample)))
     z_p = 2 * (1 - norm.cdf(abs(z_stat)))

     t_stat, t_p = ttest_1samp(sample, 105)

     print("Z-test:", z_stat, z_p)
     print("T-test:", t_stat, t_p)
```

```
Z-test: -0.5477493624875283 0.5838640127016492
T-test: -0.8317242083295219 0.42708043794566763
```

Q.No.9 Write a Python function to calculate the confidence interval for a sample mean and explain its significance.
Ans-

```
def confidence_interval(data, confidence=0.95):
    mean = np.mean(data)
    std_err = np.std(data, ddof=1) / np.sqrt(len(data))
    margin = norm.ppf((1 + confidence) / 2) * std_err
    return mean - margin, mean + margin

data = np.random.normal(100, 10, 40)
ci = confidence_interval(data)
print("Confidence Interval:", ci)
```

```
Confidence Interval: (np.float64(100.74004598432762), np.float64(106.23749933874356))
```

Q.No.10 Write a Python program to calculate the margin of error for a given confidence level using sample data.
Ans-

```
[10] confidence = 0.95
     data = np.random.normal(50, 5, 30)
     mean = np.mean(data)
     std_err = np.std(data, ddof=1) / np.sqrt(len(data))
     z_score = norm.ppf((1 + confidence) / 2)
     margin_error = z_score * std_err
     print("Margin of Error:", margin_error)
```

```
Margin of Error: 1.9382926899351973
```

Q.No.11 Implement a Bayesian inference method using Bayes' Theorem in Python and explain the process.
Ans-

```
[11]
     P_A = 0.01
     P_not_A = 0.99
     P_B_given_A = 0.95
     P_B_given_not_A = 0.10

     P_B = P_B_given_A * P_A + P_B_given_not_A * P_not_A
     P_A_given_B = (P_B_given_A * P_A) / P_B

     print("Posterior Probability (P(A|B)):", P_A_given_B)
```

```
Posterior Probability (P(A|B)): 0.08755760368663594
```

Q.No.12 Perform a Chi-square test for independence between two categorical variables in Python.
Ans-

```
[12] import pandas as pd
     from scipy.stats import chi2_contingency

     data = pd.DataFrame([[20, 30], [15, 35]])
     chi2, p, dof, expected = chi2_contingency(data)
     print("Chi-square Stat:", chi2)
     print("P-value:", p)
```

```
Chi-square Stat: 0.7032967032967032
P-value: 0.4016781664697727
```

Q.No.13 Write a Python program to calculate the expected frequencies for a Chi-square test based on observed data.
Ans-

```
[13] observed = np.array([[20, 30], [15, 35]])
     _, _, _, expected = chi2_contingency(observed)
     print("Expected Frequencies:\n", expected)
```

```
Expected Frequencies:
 [[17.5 32.5]
 [17.5 32.5]]
```

Q.No.14 Perform a goodness-of-fit test using Python to compare the observed data to an expected distribution.
Ans-

```
from scipy.stats import chisquare

observed = [18, 22, 30]
expected = [20, 20, 30]

chi2, p = chisquare(f_obs=observed, f_exp=expected)
print("Chi-square Statistic:", chi2)
print("P-value:", p)
```

```
Chi-square Statistic: 0.4
P-value: 0.8187307530779818
```