1.Generate a list of 100 integers containing values between 90 to 130 and store it in the variable `int_list`.

After generating the list, find the following:

Explanation with Example

take a small list:

```
data = [5, 7, 7, 9, 10]
```

1. Mean

Formula:

Mean=Sum of valuesCount|text{Mean} = |frac{|text{Sum of values}}{|text{Count}}

Sum = 5+7+7+9+10 = 38

Count = 5

Mean = 38 / 5 = **7.6**

2. Median

- Sort the list → `[5, 7, 7, 9, 10]`
- Middle value = **7** (since count is odd)

If even count → take the average of the two middle values.

3. Mode

Most frequent value(s).

Here, **7** occurs twice → Mode = 7, Frequency = 2.

4. Weighted Mean

Some values are more important (weights).

Example:

Values = [50, 60, 80], Weights = [2, 1, 3]

Weighted Mean = (50×2 + 60×1 + 80×3) / (2+1+3)

= (100 + 60 + 240) / 6 = **66.67**

5. Geometric Mean

GM=x1×x2×…×xnnGM = \sqrt[n]{x_1 \times x_2 \times … \times x_n}

For `[4, 16]` → GM = √(4×16) = √64 = **8**.

6. Harmonic Mean

HM=n∑(1/x)HM = \frac{n}{\sum (1/x)}

For `[60, 40]`:

HM = 2 / (1/60 + 1/40) = 48 km/h.

7. Midrange

Midrange=Min+Max2\text{Midrange} = \frac{\text{Min} + \text{Max}}{2}

For `[5, 7, 7, 9, 10]`: (5 + 10)/2 = **7.5**.

8. Trimmed Mean

Remove some smallest & largest values, then find mean.

Example: `[5, 7, 7, 9, 100]` → Remove top 10% and bottom 10% (1 value each side) → `[7, 7, 9]` → Mean = **7.67**.

2. Generate a list of 500 integers containing values between 200 to 300 and store it in the variable `int_list2`. After generating the list, find the following:

Ans :

```
data = [10, 12, 14, 15, 18, 20, 22]
```

actual dataset (`int_list2`)

(i) Frequency & Gaussian Distribution

- **Frequency** = how often each value (or range of values) occurs.
- **Gaussian Distribution** (Normal Curve) is a bell-shaped curve defined by mean (μ) and standard deviation (σ).

In plots:

  - Histogram → shows frequency.
  - Red curve → shows idealized Gaussian shape for same μ & σ.

(i-2) Frequency + KDE

- **KDE (Kernel Density Estimate)** = smooth version of histogram; good for continuous data.

- In the plot, the KDE is the smooth curve that “flows” instead of showing blocky histogram bars.

(i-3) Gaussian vs KDE

- Gaussian is a **theoretical** normal distribution using mean & std.
- KDE is an **empirical** smooth curve from actual data.

(ii) Range

Formula:

$$Range = Max\ value - Min\ value$$

Example: `max=22`, `min=10` → Range = 22 - 10 = **12**

(iii) Variance & Standard Deviation

- **Variance** measures how far numbers spread from mean.

Formula:

$$Variance = \sum (xi - \bar{x})2n$$

- **Standard Deviation (SD)** = √Variance.

Example:

Mean = 15.86

Variance ≈ 16.81

SD ≈ 4.10

(iv) Interquartile Range (IQR)

- Spread of the **middle 50%** of data.

Formula:

$IQR = Q3 - Q1$

Example:

Q1 = 12, Q3 = 20 → IQR = 8

(v) Coefficient of Variation (CV)

- **Relative measure** of variability (percentage).

Formula:

$CV = \frac{\text{SD}}{\text{Mean}} \times 100$

Example: CV = (4.10 / 15.86) × 100 ≈ 25.85%

(vi) Mean Absolute Deviation (MAD)

- Average of **absolute** differences from the mean.

Formula:

$MAD = \frac{\sum |x_i - \bar{x}|}{n}$

Example: MAD ≈ 3.43

(vii) Quartile Deviation

- **Half** of IQR (Semi-Interquartile Range).

Formula:

$QD=IQR2QD = \frac{IQR}{2}$

Example: QD = 8 / 2 = 4

(viii) Range-based Coefficient of Dispersion

- **Relative spread** based on range.

Formula:

$Range\ Coefficient\ of\ Dispersion=Max−MinMax+Min \text{Range Coefficient of Dispersion} = \frac{\text{Max} - \text{Min}}{\text{Max} + \text{Min}}$

Example: (22 − 10) / (22 + 10) = 12 / 32 = 0.375

3.Write a Python class representing a discrete random variable with methods to calculate its expected value and variance.

Ans

Simple Python code :

```
class DiscreteRandomVariable:
    def __init__(self, values, probabilities):
        self.values = values
        self.probabilities = probabilities

    def expected_value(self):
        return sum(v * p for v, p in zip(self.values,
self.probabilities))

    def variance(self):
```

```
        mean = self.expected_value()
        return sum((v**2) * p for v, p in zip(self.values,
self.probabilities)) - mean**2


# Example: Fair die
values = [1, 2, 3, 4, 5, 6]
probabilities = [1/6] * 6

drv = DiscreteRandomVariable(values, probabilities)

print("Expected Value:", drv.expected_value())
print("Variance:", drv.variance())
```

Output

```
Expected Value: 3.5
Variance: 2.916666666666666
```

Explanation (in simple terms)

- `expected_value()` → multiplies each value by its probability, adds them up.
- `variance()` → measures how spread out the values are from the mean.

4.Implement a program to simulate the rolling of a fair six-sided die and calculate the expected value and variance of the outcomes.

```
import random


# Roll a die N times
def roll_die(num_rolls):
    rolls = [random.randint(1, 6) for _ in range(num_rolls)]
    return rolls


# Calculate Expected Value
def expected_value(values):
    return sum(values) / len(values)


# Calculate Variance
def variance(values):
    mean = expected_value(values)
    return sum((x - mean) ** 2 for x in values) / len(values)


# Simulation
rolls = roll_die(10000)  # roll die 10,000 times
print("Expected Value:", expected_value(rolls))
print("Variance:", variance(rolls))
```

Example Output:

```
Expected Value: 3.49
Variance: 2.90
```

(Values will vary slightly because of randomness, but Expected Value ≈ **3.5**, Variance ≈ **2.92** for a fair die.)

5. Generate Random Samples from a Distribution and Calculate Mean & Variance

use **NumPy** because it has built-in functions for probability distributions like **binomial** and **Poisson**.

```
import numpy as np

def generate_distribution_samples(dist_type, size, **kwargs):
    if dist_type == "binomial":
        samples = np.random.binomial(kwargs['n'], kwargs['p'], size)
    elif dist_type == "poisson":
        samples = np.random.poisson(kwargs['lam'], size)
    else:
        raise ValueError("Unsupported distribution type!")
    return samples

def calculate_stats(samples):
    mean = np.mean(samples)
    variance = np.var(samples)
    return mean, variance

# Example: Binomial distribution (n=10 trials, p=0.5)
binomial_samples = generate_distribution_samples("binomial", 10000,
n=10, p=0.5)
mean, var = calculate_stats(binomial_samples)
print("Binomial → Mean:", mean, "Variance:", var)

# Example: Poisson distribution (λ = 4)
poisson_samples = generate_distribution_samples("poisson", 10000,
lam=4)
mean, var = calculate_stats(poisson_samples)
print("Poisson → Mean:", mean, "Variance:", var)
```

Example Output:

```
Binomial → Mean: 5.02 Variance: 2.48
Poisson → Mean: 4.00 Variance: 3.98
```

6.Write a Python script to generate random numbers from a Gaussian (normal) distribution and compute the mean, variance, and standard deviation of the samples. with easy

Python Code

```
import numpy as np

# Generate 1000 random numbers from a Gaussian (Normal) distribution
# mean = 50, standard deviation = 10
samples = np.random.normal(loc=50, scale=10, size=1000)

# Calculate statistics
mean = np.mean(samples)
variance = np.var(samples)
std_dev = np.std(samples)

# Display results
print("Mean:", mean)
print("Variance:", variance)
print("Standard Deviation:", std_dev)
```

How it works

1. **`np.random.normal(loc, scale, size)`**
    a. `loc` → Mean (center) of the distribution.
    b. `scale` → Standard deviation (spread).
    c. `size` → Number of random numbers to generate.
2. **`np.mean()`** → Calculates average of the values.
3. **`np.var()`** → Finds variance (spread squared).
4. **`np.std()`** → Finds standard deviation (spread).

Example Output:

```
Mean: 49.85
Variance: 99.12
Standard Deviation: 9.95
```

These values are close to our set mean = 50 and std dev = 10 because randomness causes small variations.

7. Use seaborn library to load `tips` dataset. Find the following from the dataset for the columns `total_bill` and `tip`:

Easy Python Code

```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt

# Load tips dataset
tips = sns.load_dataset("tips")

# (i) Skewness
print("Skewness of total_bill:", tips['total_bill'].skew())
print("Skewness of tip:", tips['tip'].skew())

# (ii) Type of skewness
if tips['total_bill'].skew() > 0:
    print("total_bill: Positive Skew")
else:
    print("total_bill: Negative Skew or Symmetric")

if tips['tip'].skew() > 0:
    print("tip: Positive Skew")
else:
    print("tip: Negative Skew or Symmetric")
```

```
# (iii) Covariance
print("Covariance:", np.cov(tips['total_bill'], tips['tip'])[0][1])

# (iv) Pearson correlation
print("Pearson correlation:", tips['total_bill'].corr(tips['tip']))

# (v) Scatter plot
plt.scatter(tips['total_bill'], tips['tip'])
plt.xlabel("Total Bill")
plt.ylabel("Tip")
plt.title("Total Bill vs Tip")
plt.show()
```

How it works (in simple words)

1. **`skew()`** → finds skewness directly.
2. **If-else** → checks if positive or not.
3. **`np.cov()`** → finds covariance.
4. **`corr()`** → finds Pearson correlation.
5. **`plt.scatter()`** → makes a scatter plot

8.Write a Python function to calculate the probability density function (PDF) of a continuous random variable for a given normal distribution.

Ans

Easy Python Code

```
import math

# Function to calculate PDF of a normal distribution
def normal_pdf(x, mean, std_dev):
    return (1 / (std_dev * math.sqrt(2 * math.pi))) *
math.exp(-((x - mean) ** 2) / (2 * std_dev ** 2))
```

```
# Example
mean = 50
std_dev = 10
x = 55

pdf_value = normal_pdf(x, mean, std_dev)
print(f"PDF at x = {x} is {pdf_value}")
```

How it works

1. **Formula** used:

$$PDF(x) = \frac{1}{\sigma \sqrt{2\pi}} \; e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

where

   a. μ | *mu* = mean
   b. σ | *sigma* = standard deviation
   c. xx = point to calculate PDF
2. **math.exp()** → calculates the exponential.
3. **math.sqrt()** → finds square root.

9.Create a program to calculate the cumulative distribution function (CDF) of exponential distribution.

Ans :

Easy Python Code

```
import math

# Function to calculate PDF of a normal distribution
```

```
def normal_pdf(x, mean, std_dev):
    return (1 / (std_dev * math.sqrt(2 * math.pi))) *
math.exp(-((x - mean) ** 2) / (2 * std_dev ** 2))

# Example
mean = 50
std_dev = 10
x = 55

pdf_value = normal_pdf(x, mean, std_dev)
print(f"PDF at x = {x} is {pdf_value}")
```

How it works

1. **Formula** used:

$$PDF(x) = \frac{1}{\sigma \sqrt{2\pi}} \; e^{-\frac{(x - \mu)^2}{2\sigma^2}}$$

where

   a. μ *mu* = mean
   b. σ *sigma* = standard deviation
   c. x *x* = point to calculate PDF
2. **math.exp()** → calculates the exponential.
3. **math.sqrt()** → finds square root.

10. Write a Python function to calculate the probability mass function (PMF) of Poisson distribution.

Ans :

Python Code

```
import math

# Function to calculate PMF of Poisson distribution
def poisson_pmf(k, lam):
    return (math.exp(-lam) * (lam ** k)) / math.factorial(k)

# Example
lam = 4   # average rate (λ)
k = 2     # number of events

pmf_value = poisson_pmf(k, lam)
print(f"PMF for k = {k} and λ = {lam} is {pmf_value}")
```

How it works

Formula for Poisson PMF:

$PMF(k)=e−λλkk!PMF(k) = \frac{e^{-\lambda} \lambda^k}{k!}$

where

- $\lambda \lambda$ = average rate of events
- kk = number of occurrences
- $ee \approx 2.718$ (math.exp handles it)

11. A company wants to test if a new website layout leads to a higher conversion rate (percentage of visitors who make a

purchase). They collect data from the old and new layouts to compare.

Python Code

```
import numpy as np
from statsmodels.stats.proportion import proportions_ztest

# Data
old_layout = np.array([1] * 50 + [0] * 950)   # 50 purchases out of
1000
new_layout = np.array([1] * 70 + [0] * 930)   # 70 purchases out of
1000

# Count of successes (purchases)
success_counts = np.array([old_layout.sum(), new_layout.sum()])

# Number of trials (total visitors)
nobs = np.array([len(old_layout), len(new_layout)])

# Apply two-proportion Z-test
stat, p_value = proportions_ztest(success_counts, nobs)

print(f"Z-statistic: {stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation at 5% significance level
if p_value < 0.05:
    print("Result: Significant difference → One layout performs
better.")
else:
    print("Result: No significant difference → Layouts perform
similarly.")
```

Explanation

- We are testing:
  - **H_0 (Null Hypothesis):** No difference in conversion rate between old and new layouts.
  - **H_1 (Alternative Hypothesis):** Conversion rate is different (or higher for new).
- **`proportions_ztest`** compares the proportions of two groups.
- **p-value < 0.05** → reject H_0 → the difference is significant.

12. A tutoring service claims that its program improves students' exam scores. A sample of students who participated in the program was taken, and their scores before and after the program were recorded.

Use the below code to generate samples of respective arrays of marks: ```python before_program = np.array([75, 80, 85, 70, 90, 78, 92, 88, 82, 87]) after_program = np.array([80, 85, 90, 80, 92, 80, 95, 90, 85, 88]) ```

Use z-test to find if the claims made by tutor are true or false.

Ans :

Python Code

```
import numpy as np
from statsmodels.stats.proportion import proportions_ztest

# Data
old_layout = np.array([1] * 50 + [0] * 950)   # 50 purchases out of
1000
new_layout = np.array([1] * 70 + [0] * 930)   # 70 purchases out of
1000

# Count of successes (purchases)
success_counts = np.array([old_layout.sum(), new_layout.sum()])
```

```
# Number of trials (total visitors)
nobs = np.array([len(old_layout), len(new_layout)])

# Apply two-proportion Z-test
stat, p_value = proportions_ztest(success_counts, nobs)

print(f"Z-statistic: {stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation at 5% significance level
if p_value < 0.05:
    print("Result: Significant difference → One layout performs
better.")
else:
    print("Result: No significant difference → Layouts perform
similarly.")
```

Explanation

- We are testing:
  - **H_0 (Null Hypothesis):** No difference in conversion rate between old and new layouts.
  - **H_1 (Alternative Hypothesis):** Conversion rate is different (or higher for new).
- **`proportions_ztest`** compares the proportions of two groups.
- **p-value < 0.05** → reject H_0 → the difference is significant.

13. A pharmaceutical company wants to determine if a new drug is effective in reducing blood pressure. They conduct a study and record blood pressure measurements before and after administering the drug.

Ans

Easy Z-Test for Paired Data

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

# Data
before_drug = np.array([145, 150, 140, 135, 155, 160, 152,
148, 130, 138])
after_drug  = np.array([130, 140, 132, 128, 145, 148, 138,
136, 125, 130])

# Step 1: Calculate differences
diff = before_drug - after_drug

# Step 2: Mean and standard deviation of differences
mean_diff = np.mean(diff)
std_diff = np.std(diff, ddof=1)  # sample standard deviation

# Step 3: Z-test calculation
n = len(diff)
z_stat = mean_diff / (std_diff / np.sqrt(n))

# Two-tailed p-value
p_value = 2 * (1 - norm.cdf(abs(z_stat)))

# Step 4: Output
print(f"Mean Difference: {mean_diff:.2f}")
print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Step 5: Interpretation
if p_value < 0.05:
    print("Result: Significant → Drug is effective in
reducing blood pressure.")
```

```
else:
    print("Result: Not significant → No strong evidence drug
works.")
```

Logic in Simple Terms

1. We calculate **difference** = before − after for each person.
2. Find **average reduction** and **spread** of reductions.
3. Compute Z-score =

$$Z = \frac{\bar{d}}{s_d / \sqrt{n}}$$

4. Compare p-value with 0.05 (5% significance level).

14. A customer service department claims that their average response time is less than 5 minutes. A sample of recent customer interactions was taken, and the response times were recorded.

Implement the below code to generate the array of response time:

python response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])
Implement z-test to find the claims made by customer service department are tru or false.

Ans

$H_0: \mu = 5 \quad \text{(average is 5 min)}$ $H_a: \mu < 5 \quad \text{(average is less than 5 min — claim)}$

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

# Data
response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5,
4.6, 4.4])

# Step 1: Sample statistics
sample_mean = np.mean(response_times)
```

```
sample_std = np.std(response_times, ddof=1)  # sample standard
deviation
n = len(response_times)

# Step 2: Population mean under H0
mu_0 = 5

# Step 3: Z-test calculation
z_stat = (sample_mean - mu_0) / (sample_std / np.sqrt(n))

# Step 4: One-tailed p-value (less than test)
p_value = norm.cdf(z_stat)

# Step 5: Output
print(f"Sample Mean: {sample_mean:.2f}")
print(f"Z-statistic: {z_stat:.4f}")
print(f"P-value: {p_value:.4f}")

# Step 6: Interpretation
if p_value < 0.05:
    print("Result: Significant → Claim is TRUE (avg response time < 5
mins).")
else:
    print("Result: Not significant → Claim is FALSE (no evidence avg <
5 mins).")
```

Explanation in Easy Words

1. Find **sample mean** and **sample standard deviation**.
2. Compare sample mean with claimed mean (5 minutes) using Z-formula:

$$Z = \frac{\bar{x} - \mu_0}{s / \sqrt{n}}$$

3. Use **one-tailed test** because claim says **less than**.
4. If **p-value < 0.05**, claim is supported.

15. A company is testing two different website layouts to see which one leads to higher click-through rates. Write a Python function to perform an A/B test analysis, including calculating the t-statistic, degrees of freedom, and p-value.

Use the following data:

python layout_a_clicks = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]

layout_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]

Ans

Two-Sample t-Test for A/B Testing

We check:

H0:μA=μB(no difference)H_0: \mu_A = \mu_B \quad (\text{no difference})
Ha:μA≠μB(click rates differ)H_a: \mu_A \neq \mu_B \quad (\text{click rates differ})

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

def ab_test(layout_a, layout_b):
    # Convert to numpy arrays
    a = np.array(layout_a) b = np.array(layout_b)

    # Step 1: Means and standard deviations
    mean_a = np.mean(a)
    mean_b = np.mean(b)
    std_a = np.std(a, ddof=1)
    std_b = np.std(b, ddof=1)

    # Step 2: Sample sizes
    n_a = len(a)
    n_b = len(b)

    # Step 3: t-statistic (Welch's t-test formula)
    t_stat = (mean_a - mean_b) / np.sqrt((std_a**2 / n_a) + (std_b**2
/ n_b))
```

```
    # Step 4: Degrees of freedom (Welch–Satterthwaite equation)
    df = ((std_a**2 / n_a) + (std_b**2 / n_b))**2 / \
         (((std_a**2 / n_a)**2 / (n_a - 1)) + ((std_b**2 / n_b)**2 /
(n_b - 1)))

    # Step 5: Two-tailed p-value
    p_value = 2 * (1 - t.cdf(abs(t_stat), df))

    # Step 6: Results
    print(f"Mean of Layout A: {mean_a:.2f}")
    print(f"Mean of Layout B: {mean_b:.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"Degrees of Freedom: {df:.2f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("Result: Significant → One layout performs better.")
    else:
        print("Result: Not significant → No clear difference.")

# Given data
layout_a_clicks = [28, 32, 33, 29, 31, 34, 30, 35, 36, 37]
layout_b_clicks = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]

# Run A/B test
ab_test(layout_a_clicks, layout_b_clicks)
```

How it Works in Simple Words

1. **Calculate mean** and **spread** (standard deviation) for both layouts.
2. Apply **Welch's t-test** formula to compare means (handles unequal variances).
3. Find **degrees of freedom** (needed for t-distribution).
4. Get **p-value** to check if difference is statistically significant.

16.A pharmaceutical company wants to determine if a new drug is more effective than an existing drug in reducing cholesterol levels. Create a program to analyze the clinical trial data and calculate the t statistic and p-value for the treatment effect.

Two-Sample t-Test for New vs Existing Drug

We check:

H0:μexisting=μnew(no difference)H_0: \mu_{\text{existing}} = \mu_{\text{new}} \quad (\text{no difference}) Ha:μnew<μexisting(new drug lowers cholesterol more)H_a: \mu_{\text{new}} < \mu_{\text{existing}} \quad (\text{new drug lowers cholesterol more})

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

def drug_effectiveness_test(existing, new):
    # Convert to numpy arrays
    existing = np.array(existing)
    new = np.array(new)
 # Step 1: Mean & Standard Deviation
    mean_existing = np.mean(existing)
    mean_new = np.mean(new)
    std_existing = np.std(existing, ddof=1)
    std_new = np.std(new, ddof=1)

    # Step 2: Sample sizes
    n_existing = len(existing)
    n_new = len(new)

    # Step 3: Welch's t-test statistic
    t_stat = (mean_existing - mean_new) / np.sqrt((std_existing**2 /
n_existing) + (std_new**2 / n_new))

    # Step 4: Degrees of freedom
    df = ((std_existing**2 / n_existing) + (std_new**2 / n_new))**2 /
\
         (((std_existing**2 / n_existing)**2 / (n_existing - 1)) +
((std_new**2 / n_new)**2 / (n_new - 1)))
```

```
    # Step 5: One-tailed p-value (since claim is "more effective" → one direction)
    p_value = 1 - t.cdf(t_stat, df)

    # Step 6: Results
    print(f"Mean Existing Drug Level: {mean_existing:.2f}")
    print(f"Mean New Drug Level: {mean_new:.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"Degrees of Freedom: {df:.2f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("✅ Significant result → New drug is more effective.")
    else:
        print("❌ Not significant → No strong evidence new drug is better.")

# Given cholesterol data
existing_drug_levels = [180, 182, 175, 185, 178, 176, 172, 184, 179, 183]
new_drug_levels = [170, 172, 165, 168, 175, 173, 170, 178, 172, 176]

# Run test
drug_effectiveness_test(existing_drug_levels, new_drug_levels)
```

How it works

1. **Calculates mean** cholesterol level for each group.
2. Uses **Welch's t-test** (good for unequal variances).
3. Since the claim is **"new drug is more effective"**, we use a **one-tailed test**.
4. **Low p-value (< 0.05)** → strong evidence that new drug reduces cholesterol more

17.A school district introduces an educational intervention program to improve math scores. Write a Python function to analyze pre- and post-intervention test scores, calculating the t-statistic and p-value to determine if the intervention had a significant impact

Ans

Python Code

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

def analyze_intervention(pre_scores, post_scores):
# Convert to NumPy arrays
    pre_scores = np.array(pre_scores)
    post_scores = np.array(post_scores)

    # Paired t-test
    t_stat, p_value = ttest_rel(post_scores, pre_scores)

    # Output results
    print(f"Mean Pre-Intervention: {np.mean(pre_scores):.2f}")
    print(f"Mean Post-Intervention: {np.mean(post_scores):.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("✅ Significant improvement → Intervention had an
impact.")
    else:
        print("❌ No significant improvement → Intervention effect not
proven.")

# Given data
pre_intervention_scores = [80, 85, 90, 75, 88, 82, 92, 78, 85, 87]
post_intervention_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]

# Run analysis
analyze_intervention(pre_intervention_scores,
post_intervention_scores)
```

How it works

1. **Paired t-test** is used because the same students took tests before and after.
2. **Null Hypothesis (H_0)**: No difference in mean scores before and after.
3. **Alternative Hypothesis (H_1)**: Mean score after intervention is different (or higher).
4. **Decision**:
    a. If **p < 0.05** → statistically significant improvement.
    b. If **p ≥ 0.05** → no evidence of improvement.
    c.

18.An HR department wants to investigate if there's a gender-based salary gap within the company. Develop a program to analyze salary data, calculate the t-statistic, and determine if there's a statistically significant difference between the average salaries of male and female employees.

Ans

Python Code

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

# Generate synthetic salary data
np.random.seed(0)  # reproducibility
male_salaries = np.random.normal(loc=50000, scale=10000, size=20)
female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

def analyze_salary_gap(male, female):
    # Perform independent t-test
    t_stat, p_value = ttest_ind(male, female)

    # Display results
    print(f"Mean Male Salary: {np.mean(male):.2f}")
    print(f"Mean Female Salary: {np.mean(female):.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("✅ Significant salary difference → Possible gender pay
```

```
gap.")
    else:
        print("❌ No significant salary difference detected.")

# Run the analysis
analyze_salary_gap(male_salaries, female_salaries)
```

Explanation

- **Independent t-test** is used because **male and female salaries** come from different people.
- **Null Hypothesis (H_0)**: Mean male salary = Mean female salary.
- **Alternative Hypothesis (H_1)**: Means are different → salary gap exists.
- **Decision Rule**:
  - $p < 0.05$ → reject H_0 → salary gap statistically significant.
  - $p \geq 0.05$ → fail to reject H_0 → no evidence of salary gap.

19.A manufacturer produces two different versions of a product and wants to compare their quality scores. Create a Python function to analyze quality assessment data, calculate the t-statistic, and decide whether there's a significant difference in quality between the two versions.

Ans

Python Code

```
from scipy.stats import ttest_ind

# Given data
version1_scores = [85, 88, 82, 89, 87, 84, 90, 88, 85, 86, 91, 83, 87,
84, 89, 86, 84, 88, 85, 86, 89, 90, 87, 88, 85]
version2_scores = [80, 78, 83, 81, 79, 82, 76, 80, 78, 81, 77, 82, 80,
```

```
79, 82, 79, 80, 81, 79, 82, 79, 78, 80, 81, 82]

def analyze_quality_scores(v1, v2):
    # Perform independent t-test
    t_stat, p_value = ttest_ind(v1, v2)

    # Display results
    print(f"Mean Quality (Version 1): {sum(v1)/len(v1):.2f}")
    print(f"Mean Quality (Version 2): {sum(v2)/len(v2):.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("✅ Significant difference in quality between Version 1
and Version 2.")
    else:
        print("❌ No significant difference in quality between the
versions.")

# Run the analysis
analyze_quality_scores(version1_scores, version2_scores)
```

How It Works

- **Independent t-test** is used because **Version 1 and Version 2 scores** are from different product batches.
- **H_0 (Null Hypothesis)** → Mean quality of both versions is equal.
- **H_1 (Alternative Hypothesis)** → Mean quality is different.
- **$p < 0.05$** → Reject H_0 → Significant quality difference.
- **$p \geq 0.05$** → Fail to reject H_0 → No significant difference.

20. A restaurant chain collects customer satisfaction scores for two different branches. Write a program to analyze the scores, calculate the t-statistic, and determine if there's a statistically significant difference in customer satisfaction between the branches.

Ans

Python Code

```
from scipy.stats import ttest_ind

# Given data
branch_a_scores = [4, 5, 3, 4, 5, 4, 5, 3, 4, 4, 5, 4, 4, 3, 4, 5, 5,
4, 3, 4, 5, 4, 3, 5, 4, 4, 5, 3, 4, 5, 4]
branch_b_scores = [3, 4, 2, 3, 4, 3, 4, 2, 3, 3, 4, 3, 3, 2, 3, 4, 4,
3, 2, 3, 4, 3, 2, 4, 3, 3, 4, 2, 3, 4, 3]

def analyze_satisfaction(branch_a, branch_b):
    # Perform independent t-test
    t_stat, p_value = ttest_ind(branch_a, branch_b)

    # Display results
    print(f"Mean Satisfaction (Branch A):
{sum(branch_a)/len(branch_a):.2f}")
    print(f"Mean Satisfaction (Branch B):
{sum(branch_b)/len(branch_b):.2f}")
    print(f"T-statistic: {t_stat:.4f}")
    print(f"P-value: {p_value:.4f}")

    if p_value < 0.05:
        print("✅ Significant difference in customer satisfaction
between Branch A and Branch B.")
    else:
        print("❌ No significant difference in customer satisfaction
between the branches.")

# Run the analysis
analyze_satisfaction(branch_a_scores, branch_b_scores)
```

Explanation

- **Independent t-test** → Used for comparing means from **two independent groups**.
- $\mathbf{H_0}$: No difference in customer satisfaction.
- $\mathbf{H_1}$: There is a difference in customer satisfaction.

20.A political analyst wants to determine if there is a significant association between age groups and voter preferences (Candidate A or Candidate B). They collect data from a sample of 500 voters and classify them into different age groups and candidate preferences. Perform a Chi-Square test to determine if there is a significant association between age groups and voter preferences.

Ans

Python Code

```
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

# Generate synthetic data
np.random.seed(0)
age_groups = np.random.choice(['18-30', '31-50', '51+'], size=30)
voter_preferences = np.random.choice(['Candidate A', 'Candidate B'],
size=30)

# Create a DataFrame
df = pd.DataFrame({
    'Age Group': age_groups,
    'Voter Preference': voter_preferences
})

# Create a contingency table
contingency_table = pd.crosstab(df['Age Group'], df['Voter
Preference'])
```

```
print("Contingency Table:")
print(contingency_table)

# Perform Chi-Square Test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Display results
print("\nChi-Square Test Results:")
print(f"Chi-square Statistic: {chi2:.4f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p:.4f}")
print("\nExpected Frequencies:")
print(pd.DataFrame(expected, index=contingency_table.index,
columns=contingency_table.columns))

# Decision
if p < 0.05:
    print("\n✅ Significant association between Age Group and Voter
Preference.")
else:
    print("\n❌ No significant association between Age Group and Voter
Preference.")
```

Explanation

- **Chi-Square Test of Independence** → Checks if two categorical variables are related.
- **H_0**: Age group and voter preference are independent.
- **H_1**: Age group and voter preference are dependent (associated).
- **Decision Rule**:
    - **p < 0.05** → Reject H_0 → Significant association exists.
    - **p ≥ 0.05** → Fail to reject H_0 → No association found.

22. A company conducted a customer satisfaction survey to determine if there is a significant relationship between product satisfaction levels (Satisfied, Neutral, Dissatisfied) and the region where customers are located (East, West, North, South). The survey data is summarized in a contingency table.

Conduct a Chi Square test to determine if there is a significant relationship between product satisfaction levels and customer regions.

#Sample data: Product satisfaction levels (rows) vs.

Customer regions (columns) data = np.array([[50, 30, 40, 20], [30, 40, 30, 50], [20, 30, 40, 30]])

Ans

Here's the **easy Python code** to solve Question 22 using the **Chi-Square test of independence**:

Python Code

```
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

# Sample data: rows = Satisfaction levels, columns = Regions
data = np.array([
    [50, 30, 40, 20],  # Satisfied
    [30, 40, 30, 50],  # Neutral
    [20, 30, 40, 30]   # Dissatisfied
])

# Define row and column labels
row_labels = ['Satisfied', 'Neutral', 'Dissatisfied']
col_labels = ['East', 'West', 'North', 'South']

# Create a DataFrame for better readability
```

```
contingency_table = pd.DataFrame(data, index=row_labels,
columns=col_labels)
print("Contingency Table:")
print(contingency_table)

# Perform Chi-Square Test
chi2, p, dof, expected = chi2_contingency(data)

# Display results
print("\nChi-Square Test Results:")
print(f"Chi-square Statistic: {chi2:.4f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p:.4f}")

print("\nExpected Frequencies:")
print(pd.DataFrame(expected, index=row_labels, columns=col_labels))

# Decision
if p < 0.05:
    print("\n✅ Significant relationship between Product Satisfaction
and Region.")
else:
    print("\n❌ No significant relationship between Product
Satisfaction and Region.")
```

Explanation

- **Chi-Square Test of Independence** checks if two categorical variables are related.
- **H_0 (Null Hypothesis)** → No relationship between satisfaction level and customer region.
- **H_1 (Alternative Hypothesis)** → Relationship exists between satisfaction level and customer region.
- **p < 0.05** → Reject H_0 → Significant relationship exists.
- **p ≥ 0.05** → Fail to reject H_0 → No relationship found.

23. A company implemented an employee training program to improve job performance (Effective, Neutral, Ineffective). After the training, they collected data from a sample of employees and classified them based on their job performance before and after the training. Perform a Chi-Square test to determine if there is a significant difference between job performance levels before and after the training.

Ans

Python Code

```
import numpy as np
import pandas as pd
from scipy.stats import chi2_contingency

# Sample data: rows = Job performance before, columns = Job
performance after
data = np.array([
    [50, 30, 20],  # Before: Effective
    [30, 40, 30],  # Before: Neutral
    [20, 30, 40]   # Before: Ineffective
])

# Labels
row_labels = ['Before Effective', 'Before Neutral', 'Before
Ineffective']
col_labels = ['After Effective', 'After Neutral', 'After Ineffective']

# Create DataFrame for readability
contingency_table = pd.DataFrame(data, index=row_labels,
columns=col_labels)
print("Contingency Table:")
print(contingency_table)

# Chi-Square Test
chi2, p, dof, expected = chi2_contingency(data)

# Display results
```

```
print("\nChi-Square Test Results:")
print(f"Chi-square Statistic: {chi2:.4f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-value: {p:.4f}")

print("\nExpected Frequencies:")
print(pd.DataFrame(expected, index=row_labels, columns=col_labels))

# Decision
if p < 0.05:
    print("\n✅ Significant difference in job performance before and after training.")
else:
    print("\n❌ No significant difference in job performance before and after training.")
```

Explanation

- **H_0 (Null Hypothesis)** → No difference in job performance levels before vs. after training.
- **H_1 (Alternative Hypothesis)** → There is a difference in job performance before vs. after training.
- **Decision Rule**:
    - If **p < 0.05** → Reject H_0 → Training had a significant impact.
    - **If p ≥ 0.05 → Fail to reject H_0 → No evidence training changed performance.**

24. A company produces three different versions of a product: Standard, Premium, and Deluxe. The company wants to determine if there is a significant difference in customer satisfaction scores among the three product versions. They conducted a survey and collected customer satisfaction scores for each version from a random sample of customers.

Perform an ANOVA test to determine if there is a significant difference in customer satisfaction scores.

Ans

Python Code

```
from scipy.stats import f_oneway

# Sample data
standard_scores = [80, 85, 90, 78, 88, 82, 92, 78, 85, 87]
premium_scores = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]
deluxe_scores = [95, 98, 92, 97, 96, 94, 98, 97, 92, 99]

# Perform One-Way ANOVA
f_stat, p_value = f_oneway(standard_scores, premium_scores,
deluxe_scores)

# Results
print(f"F-Statistic: {f_stat:.4f}")
print(f"P-Value: {p_value:.4f}")

# Decision
if p_value < 0.05:
    print("✅ Significant difference in satisfaction scores among the
product versions.")
else:
    print("❌ No significant difference in satisfaction scores among
the product versions.")
```

Explanation

- **H_0 (Null Hypothesis)** → Mean satisfaction scores are **equal** for Standard, Premium, and Deluxe.
- **H_1 (Alternative Hypothesis)** → At least one version has a **different** mean satisfaction score.
- **Decision Rule**:
    - If **$p < 0.05$** → Reject H_0 → At least one group's mean is different.
    - If **$p \geq 0.05$** → Fail to reject H_0 → No significant difference found.