Q1. 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: (i) Write a Python function to calculate the mean of a given list of numbers. Create a function to find the median of a list of numbers?

Answer =>

```
import numpy as np

np.random.seed(0)

int_list = np.random.randint(90, 131, 100)

print(int_list)

def mean_list(lst):

  return sum(lst) / len(lst)

print("Mean:", mean_list(int_list))

def median_list(lst):

  lst = sorted(lst)

  n = len(lst)

  mid = n // 2

  if n % 2 == 0:

    return (lst[mid - 1] + lst[mid]) / 2

  else:

    return lst[mid]

print("Median:", median_list(int_list))
```

(ii) Develop a program to compute the mode of a list of integers.

Answer:

```
from collections import Counter


def mode_list(lst):

  freq = Counter(lst)

  max_count = max(freq.values())

  mode_values = [k for k, v in freq.items() if v == max_count]
```

```python
    return mode_values

print("Mode:", mode_list(int_list))
```

(iii) Implement a function to calculate the weighted mean of values and weights.

Answer:

```python
import numpy as np

def weighted_mean(values, weights):
    values = np.array(values)
    weights = np.array(weights)
    return np.sum(values * weights) / np.sum(weights)

weights = np.random.randint(1, 6, 100)
print("Weighted Mean:", weighted_mean(int_list, weights))
```

(iv) Write a Python function to find the geometric mean of a list of positive numbers.

Answer:

```python
def geometric_mean(lst):
    product = 1
    for x in lst:
        product *= x
    return product ** (1 / len(lst))

print("Geometric Mean:", geometric_mean(int_list))
```

(v) Create a program to calculate the harmonic mean of a list of values.

Answer:

```
def harmonic_mean(lst):
    return len(lst) / sum(1/x for x in lst)

print("Harmonic Mean:", harmonic_mean(int_list))
```

(vi) Build a function to determine the midrange of a list of numbers.

Answer:

```
def midrange(lst):
    return (min(lst) + max(lst)) / 2

print("Midrange:", midrange(int_list))
```

(vii) Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of outliers.

Answer:

```
def trimmed_mean(lst, trim_percent=0.1):
    lst = sorted(lst)
    n = len(lst)
    trim_count = int(n * trim_percent)
    trimmed_lst = lst[trim_count : n - trim_count]
    return sum(trimmed_lst) / len(trimmed_lst)

print("Trimmed Mean (10%):", trimmed_mean(int_list, 0.1))
```

Q2. Generate int_list2 (500 integers between 200 to 300).

Answer:

```
import numpy as np
np.random.seed(0)
```

int_list2 = np.random.randint(200, 301, 500)

print(int_list2)

i) Compare the given list of visualization for the given data

---

1. Frequency & Gaussian distribution

Answer:

```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
plt.figure(figsize=(7,5))
sns.histplot(int_list2, bins=25, stat="density")
x = np.linspace(min(int_list2), max(int_list2), 200)
plt.plot(x, stats.norm.pdf(x, np.mean(int_list2), np.std(int_list2)), linewidth=2)
plt.title("Frequency & Gaussian Distribution")
plt.show()
```

---

2. Frequency smoothened KDE plot

Answer:

```
plt.figure(figsize=(7,5))
sns.histplot(int_list2, bins=25, kde=True, stat="density")
plt.title("Frequency Smoothened KDE Plot")
plt.show()
```

3. Gaussian distribution & smoothened KDE plot

Answer:

```
plt.figure(figsize=(7,5))
sns.kdeplot(int_list2, linewidth=2)
x = np.linspace(min(int_list2), max(int_list2), 200)
plt.plot(x, stats.norm.pdf(x, np.mean(int_list2), np.std(int_list2)), linewidth=2)
```

```python
plt.title("Gaussian Distribution & Smoothened KDE Plot")
plt.show()
```

(ii) Write a Python function to calculate the range of a given list of numbers.

Answer:

```python
def data_range(lst):
    return max(lst) - min(lst)
print("Range:", data_range(int_list2))
```

(iii) Create a program to find the variance and standard deviation of a list of numbers.

Answer:

```python
def variance_std(lst):
    arr = np.array(lst)
    variance = np.var(arr, ddof=1)
    std_dev = np.std(arr, ddof=1)
    return variance, std_dev


var, std = variance_std(int_list2)
print("Variance:", var)
print("Standard Deviation:", std)
```

(iv) Implement a function to compute the interquartile range (IQR) of a list of values.

Answer:

```python
def iqr(lst):
    q1 = np.percentile(lst, 25)
    q3 = np.percentile(lst, 75)
    return q3 - q1
print("IQR:", iqr(int_list2))
```

(v) Build a program to calculate the coefficient of variation for a dataset.

Answer:

```
def coefficient_variation(lst):
    arr = np.array(lst)
    return (np.std(arr, ddof=1) / np.mean(arr)) * 100
print("Coefficient of Variation (%):", coefficient_variation(int_list2))
```

(vi) Write a Python function to find the mean absolute deviation (MAD) of a list of numbers.

Answer:

```
def mad(lst):
    mean_val = np.mean(lst)
    return np.mean([abs(x - mean_val) for x in lst])
print("MAD:", mad(int_list2))
```

(vii) Create a program to calculate the quartile deviation of a list of values.

Answer:

```
def quartile_deviation(lst):
    q1 = np.percentile(lst, 25)
    q3 = np.percentile(lst, 75)
    return (q3 - q1) / 2
print("Quartile Deviation:", quartile_deviation(int_list2))
```

(viii) Implement a function to find the range-based coefficient of dispersion for a dataset.

Answer:

```
def range_dispersion(lst):
    return (max(lst) - min(lst)) / (max(lst) + min(lst))
print("Range-based Coefficient of Dispersion:", range_dispersion(int_list2))
```

Q3. Write a Python class representing a discrete random variable with methods to calculate its expected value and variance.

Answer:

```
class DiscreteRandomVariable:
    def __init__(self, values, probabilities):
        self.values = np.array(values)
        self.probabilities = np.array(probabilities)
    def expected_value(self):
        return np.sum(self.values * self.probabilities)
    def variance(self):
        mu = self.expected_value()
        return np.sum(((self.values - mu) ** 2) * self.probabilities)
drv = DiscreteRandomVariable([1, 2, 3], [0.2, 0.5, 0.3])
print("Expected Value:", drv.expected_value())
print("Variance:", drv.variance())
```

Q4. Implement a program to simulate rolling of a fair six-sided die and calculate expected value and variance.

Answer:

```
np.random.seed(0)
die_rolls = np.random.randint(1, 7, 1000)
print("Expected Value:", np.mean(die_rolls))
print("Variance:", np.var(die_rolls, ddof=1))
```

Q5. Create a Python function to generate random samples from a distribution (Binomial/Poisson) and calculate mean and variance.

Answer:

```
def generate_samples(distribution="binomial", size=1000):
    if distribution == "binomial":
        samples = np.random.binomial(n=10, p=0.5, size=size)
    elif distribution == "poisson":
        samples = np.random.poisson(lam=5, size=size)
```

```
    else:
        return None
    return samples, np.mean(samples), np.var(samples, ddof=1)
samples, mean_val, var_val = generate_samples("poisson", 1000)
print("Mean:", mean_val)
print("Variance:", var_val)
```

Q6. Write a script to generate random numbers from Gaussian distribution and compute mean, variance and standard deviation.

Answer:

```
np.random.seed(0)
normal_samples = np.random.normal(loc=0, scale=1, size=1000)
print("Mean:", np.mean(normal_samples))
print("Variance:", np.var(normal_samples, ddof=1))
print("Standard Deviation:", np.std(normal_samples, ddof=1))
```

Q7. Use seaborn library to load tips dataset. Find the following for the columns total_bill and tip:

Q(i). Write a Python function that calculates their skewness.

Answer:

```
import seaborn as sns
from scipy import stats
tips = sns.load_dataset("tips")
def skewness(column):
    return stats.skew(column)
print("Skewness of total_bill:", skewness(tips["total_bill"]))
print("Skewness of tip:", skewness(tips["tip"]))
```

Q(ii). Create a program that determines whether the columns exhibit positive skewness, negative skewness, or approximately symmetric.

Answer:

```
def skew_type(value):
```

```
    if value > 0:
        return "Positive Skewness"
    elif value < 0:
        return "Negative Skewness"
    else:
        return "Approximately Symmetric"
print("total_bill:", skew_type(skewness(tips["total_bill"])))
print("tip:", skew_type(skewness(tips["tip"])))
```

Q(iii). Write a function that calculates the covariance between two columns.

Answer:

```
def covariance(col1, col2):
    return np.cov(col1, col2, ddof=1)[0, 1]
print("Covariance between total_bill and tip:", covariance(tips["total_bill"], tips["tip"]))
```

Q(iv). Implement a Python program that calculates the Pearson correlation coefficient between two columns.

Answer:

```
def pearson_corr(col1, col2):
    return np.corrcoef(col1, col2)[0, 1]
print("Pearson correlation between total_bill and tip:", pearson_corr(tips["total_bill"], tips["tip"]))
```

Q(v). Write a script to visualize the correlation between two specific columns using scatter plot.

Answer:

```
import matplotlib.pyplot as plt
plt.figure(figsize=(7,5))
plt.scatter(tips["total_bill"], tips["tip"])
plt.title("Scatter Plot: total_bill vs tip")
plt.xlabel("total_bill")
```

```
plt.ylabel("tip")
plt.show()
```

Q8. Write a Python function to calculate the probability density function (PDF) of a continuous random variable for a given normal distribution.

Answer:

```
import numpy as np
def normal_pdf(x, mu=0, sigma=1):
    pdf = (1 / (sigma * np.sqrt(2 * np.pi))) * np.exp(-0.5 * ((x - mu) / sigma) ** 2)
    return pdf
print("Normal PDF at x=1:", normal_pdf(1, mu=0, sigma=1))
```

Q9. Create a program to calculate the cumulative distribution function (CDF) of exponential distribution.

Answer:

```
import numpy as np
def exponential_cdf(x, lam=1):
    cdf = 1 - np.exp(-lam * x)
    return cdf
print("Exponential CDF at x=2:", exponential_cdf(2, lam=1))
```

Q10. Write a Python function to calculate the probability mass function (PMF) of Poisson distribution.

Answer:

```
import numpy as np
import math
def poisson_pmf(k, lam=5):
    pmf = (lam ** k * np.exp(-lam)) / math.factorial(k)
    return pmf
print("Poisson PMF at k=3:", poisson_pmf(3, lam=5))
```

Q11.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?

To generate the data use the following command: ``` python

import numpy as np

50 purchases out of 1000 visitors old_layout = np.array([1] * 50 + [0] * 950)

70 purchases out of 1000 visitors

new_layout = np.array([1] * 70 + [0] * 930)

``` Apply z-test to find which layout is successful.

**ANSWER:**

```
numpy as np

from statsmodels.stats.proportion import proportions_ztest

50 purchases out of 1000 visitors

old_layout = np.array([1] * 50 + [0] * 950)

70 purchases out of 1000 import visitors

new_layout = np.array([1] * 70 + [0] * 930)

Count successes

count_old = old_layout.sum()

count_new = new_layout.sum()

Total observations

n_old = len(old_layout)

n_new = len(new_layout)

Apply Z-test

z_stat, p_val = proportions_ztest([count_old, count_new], [n_old, n_new])

print("Z-statistic:", z_stat)

print("P-value:", p_val)

if p_val < 0.05:

   print("Conclusion: NEw layout is successful (higher conversion rate).")

else:
```
```

print("Conclusion: No significant difference between old and new layout.")

Q12. 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.

Answer:

```
import numpy as np

from scipy import stats

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])

# Differences

diff = after_program - before_program

# One-sample z-test on differences (assuming population std from sample)

mean_diff = np.mean(diff)

std_diff = np.std(diff, ddof=1)

n = len(diff)

z_stat = mean_diff / (std_diff / np.sqrt(n))

p_val = 1 - stats.norm.cdf(z_stat)   # one-tailed test (improvement)

print("Z-statistic:", z_stat)

print("P-value:", p_val)

if p_val < 0.05:

   print("Conclusion: Tutor claim is TRUE (scores improved).")

else:

   print("Conclusion: Tutor claim is FALSE (no significant improvement).")
```

Q13. 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?

Use the below code to generate samples of respective arrays of blood pressure: ``` python 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]) ``` Implement z-test to find if the drug really works or not.

ANSWER:

```
import numpy as np

from scipy import stats

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])

# Differences (before - after) because reduction is expected

diff = before_drug - after_drug

# One-sample z-test on differences

mean_diff = np.mean(diff)

std_diff = np.std(diff, ddof=1)

n = len(diff)

z_stat = mean_diff / (std_diff / np.sqrt(n))

p_val = 1 - stats.norm.cdf(z_stat)   # one-tailed test (reduction)

print("Z-statistic:", z_stat)

print("P-value:", p_val)

if p_val < 0.05:

    print("Conclusion: Drug works (blood pressure reduced).")

else:

    print("Conclusion: Drug does not work (no significant reduction).")
```

Q14. 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

ANSWER:

```
import numpy as np

from scipy import stats

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

# Hypothesis:

# H0: mean >= 5

# H1: mean < 5

sample_mean = np.mean(response_times)

sample_std = np.std(response_times, ddof=1)

n = len(response_times)

# Z-test statistic

z_stat = (sample_mean - 5) / (sample_std / np.sqrt(n))

# One-tailed p-value (left tail)

p_val = stats.norm.cdf(z_stat)

print("Sample Mean:", sample_mean)

print("Z-statistic:", z_stat)

print("P-value:", p_val)

if p_val < 0.05:

    print("Conclusion: Claim is TRUE (average response time < 5 minutes).")

else:

    print("Conclusion: Claim is FALSE (average response time is not less than 5 minutes).")
```

Q15. 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?

ANSWER:

```
import numpy as np

from scipy import stats

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]
```

```
def ab_test_ttest(a, b):
    a = np.array(a)
    b = np.array(b)
    # Welch’s t-test (better when variances may differ)
    t_stat, p_val = stats.ttest_ind(a, b, equal_var=False)
    # Degrees of freedom (Welch-Satterthwaite approximation)
    s1 = np.var(a, ddof=1)
    s2 = np.var(b, ddof=1)
    n1 = len(a)
    n2 = len(b)
dof = (s1/n1 + s2/n2)**2 / (((s1/n1)**2)/(n1-1) + ((s2/n2)**2)/(n2-1))
    return t_stat, dof, p_val
t_stat, dof, p_val = ab_test_ttest(layout_a_clicks, layout_b_clicks)
print("T-statistic:", t_stat)
print("Degrees of freedom:", dof)
print("P-value:", p_val)
```

Q16.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 tstatistic and p-value for the treatment effect?

Use the following data of cholestrol level: ` ` ` python 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]

ANSWER:

```
import numpy as np
from scipy import stats
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]
# Convert to numpy arrays
```

```
existing = np.array(existing_drug_levels)

new = np.array(new_drug_levels)

# Welch's t-test (does not assume equal variance)

t_stat, p_two_tailed = stats.ttest_ind(new, existing, equal_var=False)

# Convert to one-tailed p-value (since we test new < existing)

p_one_tailed = p_two_tailed / 2 if t_stat < 0 else 1 - (p_two_tailed / 2)

print("t-statistic =", t_stat)

print("p-value (one-tailed) =", p_one_tailed)
```

Q17. 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.

Use the following data of test score: ` ` ` python 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]

ANSWER:

```
import numpy as np

from scipy import stats

def intervention_ttest(pre_scores, post_scores):

   pre = np.array(pre_scores)

   post = np.array(post_scores)

   # Paired t-test

   t_stat, p_value = stats.ttest_rel(post, pre)

   return t_stat, p_value

# 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]

t_stat, p_val = intervention_ttest(pre_intervention_scores, post_intervention_scores)

print("t-statistic =", t_stat) ,print("p-value =", p_val)
```

Q18. 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.

Use the below code to generate synthetic data: ``` python # Generate synthetic salary data for male and female employees np.random.seed(0) # For reproducibility male_salaries = np.random.normal(loc=50000, scale=10000, size=20) female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

ANSWER:

```
import numpy as np

from scipy import stats

# Generate synthetic salary data for male and female employees

np.random.seed(0)  # For reproducibility

male_salaries = np.random.normal(loc=50000, scale=10000, size=20)

female_salaries = np.random.normal(loc=55000, scale=9000, size=20)

# Welch's t-test (recommended when variances may differ)

t_stat, p_value = stats.ttest_ind(male_salaries, female_salaries, equal_var=False)

print("Male salaries mean   :", np.mean(male_salaries))

print("Female salaries mean :", np.mean(female_salaries))

print("t-statistic          :", t_stat)

print("p-value              :", p_value)

# Decision (alpha = 0.05)

alpha = 0.05

if p_value < alpha:

    print("Statistically significant difference in average salaries.")

else:

    print("No statistically significant difference in average salaries.")
```

Q19. 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.

Use the following data: ` ` ` python 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]

ANSWER:

```
import numpy as np

from scipy import stats

def compare_quality(version1_scores, version2_scores, alpha=0.05):

  v1 = np.array(version1_scores)

  v2 = np.array(version2_scores)

  # Welch's t-test (safe option: does not assume equal variance)

  t_stat, p_value = stats.ttest_ind(v1, v2, equal_var=False)

  print("Mean Version 1 =", np.mean(v1))

  print("Mean Version 2 =", np.mean(v2))

  print("t-statistic =", t_stat)

  print("p-value =", p_value)

  if p_value < alpha:

    print(" Significant difference in quality between the two versions.")

  else:

    print("No significant difference in quality between the two versions.")

  return t_stat, p_value

# 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]

compare_quality(version1_scores, version2_scores)
```

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.

Use the below data of scores: ```python 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]

ANSWER:

```
import numpy as np

from scipy import stats

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,]()

a = np.array(branch_a_scores)

b = np.array(branch_b_scores)

# Welch's t-test (best for real-world data)

t_stat, p_value = stats.ttest_ind(a, b, equal_var=False)

print("Branch A Mean =", np.mean(a))

print("Branch B Mean =", np.mean(b))

print("t-statistic   =", t_stat)

print("p-value       =", p_value)

# Decision

alpha = 0.05

if p_value < alpha:

    print(" Statistically significant difference in satisfaction between branches.")

else:

    print(" No statistically significant difference in satisfaction between branches.")
```

Q21. 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?

Use the below code to generate data: ``` python np.random.seed(0) age_groups = np.random.choice([18 30 , 31 50 , 51+', 51+'], size=30) voter_preferences = np.random.choice(['Candidate A', 'Candidate B'], size=30)

ANSWER:

```
import numpy as np

import pandas as pd

from scipy.stats import chi2_contingency

np.random.seed(0)

# Generate synthetic data (fixed age group labels)

age_groups = np.random.choice(['18-30', '31-50', '51+'], size=30)

voter_preferences = np.random.choice(['Candidate A', 'Candidate B'], size=30)

# Create contingency table

data = pd.crosstab(age_groups, voter_preferences)

print("Contingency Table:\n", data)

# Chi-square test

chi2, p_value, dof, expected = chi2_contingency(data)

print("\nChi-square statistic =", chi2)

print("Degrees of freedom   =", dof)

print("p-value            =", p_value)

# Decision

alpha = 0.05

if p_value < alpha:

   print(" Significant association between age groups and voter preference.")

else:

   print(" No significant association between age groups and voter preference.")
```

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

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

Sample data: ```python #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]])

ANSWER:

```
import numpy as np

from scipy.stats import chi2_contingency

# Sample data: Satisfaction levels (rows) vs Regions (columns)

# Rows: [Satisfied, Neutral, Dissatisfied]

# Cols: [East, West, North, South]

data = np.array([

    [50, 30, 40, 20],

    [30, 40, 30, 50],

    [20, 30, 40, 30]

])

# Chi-square test

chi2, p_value, dof, expected = chi2_contingency(data)

print("Chi-square statistic =", chi2)

print("Degrees of freedom   =", dof)

print("p-value              =", p_value)

print("\nExpected Frequencies:\n", expected)

# Decision

alpha = 0.05

if p_value < alpha:

    print("\n  Significant relationship between satisfaction level and region.")

else:

    print("\n No significant relationship between satisfaction level and region.")
```

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.

Sample data: ``` python # Sample data: Job performance levels before (rows) and after (columns) training data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])

ANSWER:

```
import numpy as np

from scipy.stats import chi2_contingency

# Sample data: performance levels before (rows) and after (columns)

# Rows:    [Effective, Neutral, Ineffective]  (Before)

# Columns: [Effective, Neutral, Ineffective]  (After)

data = np.array([

   [50, 30, 20],

   [30, 40, 30],

   [20, 30, 40]

])

# Chi-square test

chi2, p_value, dof, expected = chi2_contingency(data)

print("Chi-square statistic =", chi2)

print("Degrees of freedom   =", dof)

print("p-value              =", p_value)

print("\nExpected Frequencies:\n", expected)

# Decision

alpha = 0.05

if p_value < alpha:

   print("\n Significant difference between performance levels before and after training.")

else:

   print("\n No significant difference between performance levels before and after training.")
```

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.

Use the following data: ``` python # Sample data: Customer satisfaction scores for each product version 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

ANSWER:

```
import numpy as np

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]

# One-way ANOVA test

f_stat, p_value = f_oneway(standard_scores, premium_scores, deluxe_scores)

print("F-statistic =", f_stat)

print("p-value     =", p_value)

# Decision

alpha = 0.05

if p_value < alpha:

    print(" Significant difference in customer satisfaction among product versions.")

else:

    print("No significant difference in customer satisfaction among product versions.")
```