adv-stat-2

November 4, 2024

1. Generate a list of 100 integers containing values between 90 to 130 and store it in the variable Sint _ 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.
(ii) Develop a program to compute the mode of a list of integers.
(iii) Implement a function to calculate the weighted mean of a list of values and their corresponding weights.
(iv) Write a Python function to find the geometric mean of a list of positive numbers.
(v) Create a program to calculate the harmonic mean of a list of values.
(vi) Build a function to determine the midrange of a list of numbers (average of the minimum and maximum).
(vii) Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of outliers.

[2]:
```python
import numpy as np
import random
from collections import Counter
from scipy.stats import trim_mean, gmean, hmean

# Generate a list of 100 integers between 90 and 130
int_list = [random.randint(90, 130) for _ in range(100)]

# Task (i): Mean
def calculate_mean(numbers):
    return sum(numbers) / len(numbers)

# Task (i): Median
def calculate_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)
    mid = n // 2
    if n % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    else:
        return sorted_numbers[mid]

# Task (ii): Mode
def calculate_mode(numbers):
```

```
    frequency = Counter(numbers)
    mode_data = frequency.most_common(1)
    return mode_data[0][0] if mode_data else None

# Task (iii): Weighted Mean
def calculate_weighted_mean(values, weights):
    weighted_sum = sum(v * w for v, w in zip(values, weights))
    return weighted_sum / sum(weights)

# Task (iv): Geometric Mean
def calculate_geometric_mean(numbers):
    return gmean(numbers)

# Task (v): Harmonic Mean
def calculate_harmonic_mean(numbers):
    return hmean(numbers)

# Task (vi): Midrange
def calculate_midrange(numbers):
    return (min(numbers) + max(numbers)) / 2

# Task (vii): Trimmed Mean
def calculate_trimmed_mean(numbers, proportion_to_cut):
    return trim_mean(numbers, proportion_to_cut)

# Running all tasks
print("Generated List:", int_list)
print("Mean:", calculate_mean(int_list))
print("Median:", calculate_median(int_list))
print("Mode:", calculate_mode(int_list))

# For weighted mean, creating a random weights list of 100 values between 1 and␣
 ↪10
weights = [random.randint(1, 10) for _ in range(100)]
print("Weighted Mean:", calculate_weighted_mean(int_list, weights))

print("Geometric Mean:", calculate_geometric_mean(int_list))
print("Harmonic Mean:", calculate_harmonic_mean(int_list))
print("Midrange:", calculate_midrange(int_list))

# Using 10% trimmed mean as an example
print("Trimmed Mean (10%):", calculate_trimmed_mean(int_list, 0.1))
```

```
Generated List: [91, 107, 124, 126, 127, 119, 97, 126, 104, 118, 121, 130, 112,
116, 119, 129, 118, 121, 98, 112, 116, 102, 116, 110, 93, 126, 101, 106, 91, 90,
95, 91, 126, 92, 126, 97, 94, 100, 125, 112, 104, 124, 98, 126, 111, 103, 104,
105, 100, 120, 96, 106, 125, 91, 90, 93, 113, 90, 130, 95, 113, 104, 118, 129,
```

```
94, 107, 97, 128, 118, 121, 97, 115, 106, 94, 100, 96, 128, 115, 101, 128, 112,
115, 110, 115, 124, 108, 114, 107, 102, 117, 98, 111, 116, 114, 113, 124, 124,
95, 127, 113]
Mean: 110.16
Median: 112.0
Mode: 126
Weighted Mean: 110.48576850094877
Geometric Mean: 109.48599024138716
Harmonic Mean: 108.80690983526839
Midrange: 110.0
Trimmed Mean (10%): 110.275
```

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

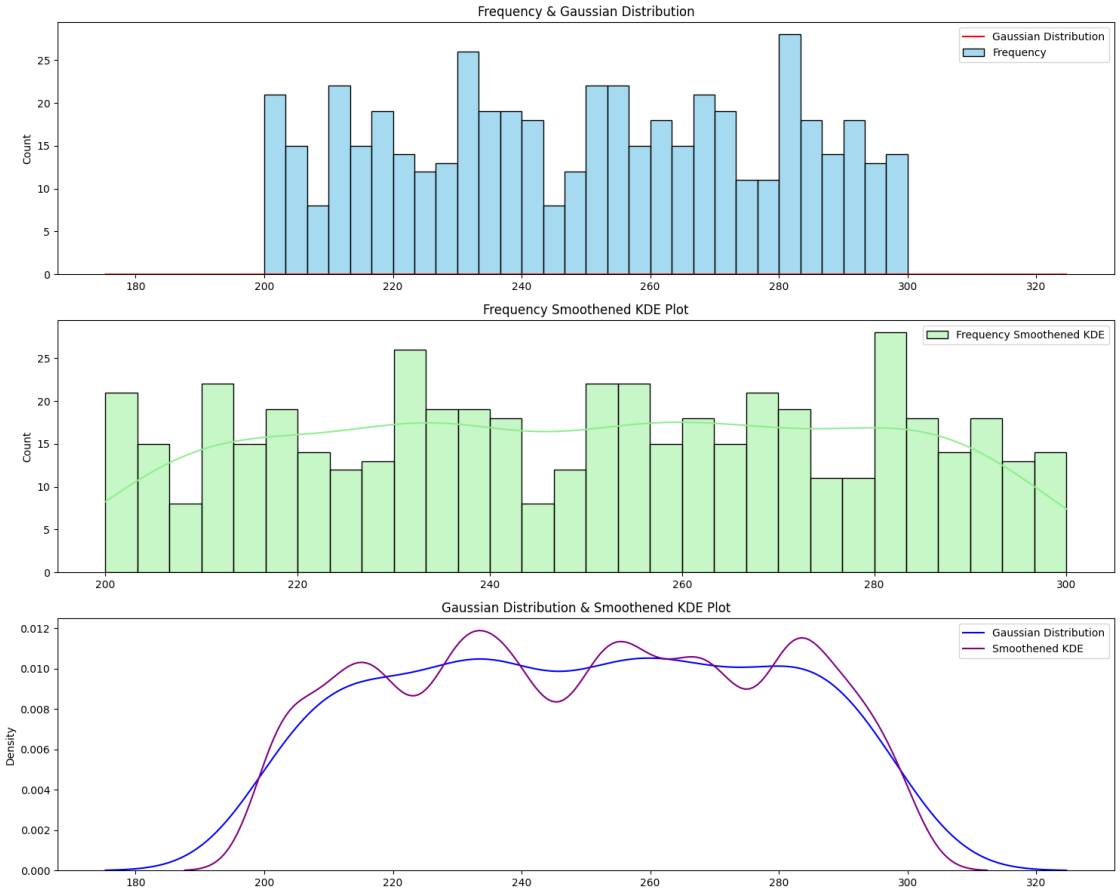
(i) Compare the given list of visualization for the given data:

1. Frequency & Gaussian distribution
2. Frequency smoothened KDE plot
3. Gaussian distribution & smoothened KDE plot

(ii) Write a Python function to calculate the range of a given list of numbers.
(iii) Create a program to find the variance and standard deviation of a list of numbers.
(iv) Implement a function to compute the interquartile range (IQR) of a list of values.
(v) Build a program to calculate the coefficient of variation for a dataset.
(vi) Write a Python function to find the mean absolute deviation (MAD) of a list of numbers.
(vii) Create a program to calculate the quartile deviation of a list of values.
(viii) Implement a function to find the range-based coefficient of dispersion for a dataset.

[4]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(15, 12))

# 1. Frequency & Gaussian distribution
plt.subplot(3, 1, 1)
sns.histplot(int_list2, bins=30, kde=False, color='skyblue', label='Frequency')
sns.kdeplot(int_list2, color='red', label='Gaussian Distribution')
plt.legend()
plt.title("Frequency & Gaussian Distribution")

# 2. Frequency smoothened KDE plot
plt.subplot(3, 1, 2)
sns.histplot(int_list2, bins=30, kde=True, color='lightgreen', label='Frequency␣
↪Smoothened KDE')
plt.legend()
plt.title("Frequency Smoothened KDE Plot")

# 3. Gaussian distribution & smoothened KDE plot
```

```
plt.subplot(3, 1, 3)
sns.kdeplot(int_list2, color='blue', label='Gaussian Distribution')
sns.kdeplot(int_list2, color='purple', bw_adjust=0.5, label='Smoothened KDE')
plt.legend()
plt.title("Gaussian Distribution & Smoothened KDE Plot")

plt.tight_layout()
plt.show()
```



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

[5]:
```
def calculate_range(data):
    return max(data) - min(data)

range_result = calculate_range(int_list2)
print("Range:", range_result)
```

```
Range: 100
```

[6]:
```
import statistics

def calculate_variance(data):
    return statistics.variance(data)

def calculate_standard_deviation(data):
    return statistics.stdev(data)

variance_result = calculate_variance(int_list2)
std_dev_result = calculate_standard_deviation(int_list2)
print("Variance:", variance_result)
print("Standard Deviation:", std_dev_result)
```

```
Variance: 812.6014068136272
Standard Deviation: 28.50616436516192
```

5. Create a Python function to generate random samples from a given probability distribution (e.g., binomial, Poisson) and calculate their mean and variance.

[7]:
```
import numpy as np

def calculate_iqr(data):
    q3, q1 = np.percentile(data, [75, 25])
    return q3 - q1

iqr_result = calculate_iqr(int_list2)
print("Interquartile Range (IQR):", iqr_result)
```

```
Interquartile Range (IQR): 48.0
```

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.

[10]:
```
def calculate_range_based_coefficient_of_dispersion(data):
    data_range = calculate_range(data)
    return data_range / (max(data) + min(data))

range_based_dispersion_result =␣
 ↪calculate_range_based_coefficient_of_dispersion(int_list2)
print("Range-Based Coefficient of Dispersion:", range_based_dispersion_result)
```

```
Range-Based Coefficient of Dispersion: 0.2
```

7. Use seaborn library to load *tips' dataset. Find the following from the dataset for the columns •total bill- and •tip•:

(i) Write a Python function that calculates their skewness.
(ii) Create a program that determines whether the columns exhibit positive skewness, negative skewness, or is approximately symmetric.
(iii) Write a function that calculates the covariance between two columns.

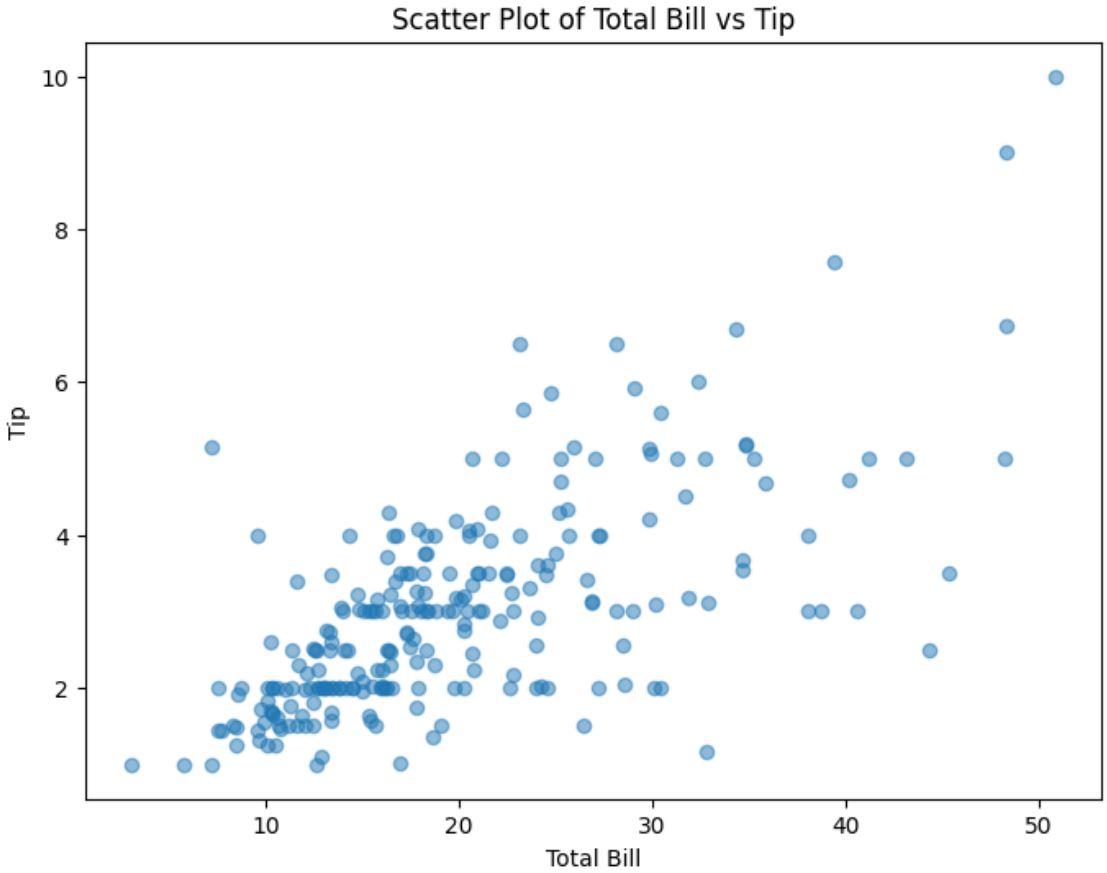
(iv) Implement a Python program that calculates the Pearson correlation coefficient between two columns.

(v) Write a script to visualize the correlation between two specific columns in a Pandas DataFrame using scatter plots.

[11]:
```
import seaborn as sns
import pandas as pd

# Load the 'tips' dataset
tips = sns.load_dataset('tips')
def calculate_skewness(column):
    return column.skew()

# Calculate skewness for 'total_bill' and 'tip'
skewness_total_bill = calculate_skewness(tips['total_bill'])
skewness_tip = calculate_skewness(tips['tip'])

print("Skewness of 'total_bill':", skewness_total_bill)
print("Skewness of 'tip':", skewness_tip)
def skewness_type(column):
    skewness = column.skew()
    if skewness > 0:
        return "Positive Skewness"
    elif skewness < 0:
        return "Negative Skewness"
    else:
        return "Approximately Symmetric"

# Determine skewness type for 'total_bill' and 'tip'
skewness_type_total_bill = skewness_type(tips['total_bill'])
skewness_type_tip = skewness_type(tips['tip'])

print("Skewness Type of 'total_bill':", skewness_type_total_bill)
print("Skewness Type of 'tip':", skewness_type_tip)
def calculate_covariance(column1, column2):
    return column1.cov(column2)

# Calculate covariance between 'total_bill' and 'tip'
covariance = calculate_covariance(tips['total_bill'], tips['tip'])
print("Covariance between 'total_bill' and 'tip':", covariance)
def calculate_pearson_correlation(column1, column2):
    return column1.corr(column2)

# Calculate Pearson correlation coefficient
pearson_corr = calculate_pearson_correlation(tips['total_bill'], tips['tip'])
print("Pearson Correlation Coefficient between 'total_bill' and 'tip':",␣
 ↪pearson_corr)
```

```python
import matplotlib.pyplot as plt

def plot_scatter(column1, column2):
    plt.figure(figsize=(8, 6))
    plt.scatter(column1, column2, alpha=0.5)
    plt.title("Scatter Plot of Total Bill vs Tip")
    plt.xlabel("Total Bill")
    plt.ylabel("Tip")
    plt.show()

# Visualize the correlation between 'total_bill' and 'tip'
plot_scatter(tips['total_bill'], tips['tip'])
```

```
Skewness of 'total_bill': 1.1332130376158205
Skewness of 'tip': 1.4654510370979401
Skewness Type of 'total_bill': Positive Skewness
Skewness Type of 'tip': Positive Skewness
Covariance between 'total_bill' and 'tip': 8.323501629224854
Pearson Correlation Coefficient between 'total_bill' and 'tip':
0.6757341092113641
```



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

[12]:
```
import math

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 usage:
mean = 0
std_dev = 1
x = 0
pdf_value = normal_pdf(x, mean, std_dev)
print("PDF of normal distribution at x=0:", pdf_value)
```

```
PDF of normal distribution at x=0: 0.3989422804014327
```

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

[13]:
```
def exponential_cdf(x, rate):
    if x < 0:
        return 0
    return 1 - math.exp(-rate * x)

# Example usage:
rate = 0.5
x = 2
cdf_value = exponential_cdf(x, rate)
print("CDF of exponential distribution at x=2:", cdf_value)
```

```
CDF of exponential distribution at x=2: 0.6321205588285577
```

1O. Write a Python function to calculate the probability mass function (PMF) of Poisson distribution.

[14]:
```
import math

def poisson_pmf(k, lam):
    return (lam ** k * math.exp(-lam)) / math.factorial(k)

# Example usage:
lam = 3
k = 2
pmf_value = poisson_pmf(k, lam)
print("PMF of Poisson distribution for k=2:", pmf_value)
```

```
PMF of Poisson distribution for k=2: 0.22404180765538775
```

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.

[15]:
```python
import numpy as np

# Data for the old layout (50 purchases out of 1000 visitors)
old_layout = np.array([1] * 50 + [0] * 950)

# Data for the new layout (70 purchases out of 1000 visitors)
new_layout = np.array([1] * 70 + [0] * 930)
from statsmodels.stats.proportion import proportions_ztest

# Define successes and observations for each layout
success_counts = [old_layout.sum(), new_layout.sum()]
total_counts = [len(old_layout), len(new_layout)]

# Perform the z-test
z_stat, p_value = proportions_ztest(success_counts, total_counts)

print("Z-Statistic:", z_stat)
print("P-Value:", p_value)

# Interpretation based on p-value
if p_value < 0.05:
    print("New layout is significantly more successful.")
else:
    print("No significant difference between layouts.")
```

```
Z-Statistic: -1.883108942886774
P-Value: 0.059685605532426224
No significant difference between layouts.
```

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: before _ program = 80, 85, 70, 90, 78, 92, 88, 82, 871) 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.

[16]:
```python
# Scores before and after the tutoring program
before_program = np.array([80, 85, 70, 90, 78, 92, 88, 82, 87])
after_program = np.array([85, 90, 80, 92, 80, 95, 90, 85, 88])
from scipy import stats

# Calculate the differences
differences = after_program - before_program
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1)
```

```
# Calculate the z-statistic
n = len(differences)
z_stat = mean_diff / (std_diff / np.sqrt(n))

# Calculate the p-value for a two-tailed test
p_value = stats.norm.sf(abs(z_stat)) * 2

print("Z-Statistic:", z_stat)
print("P-Value:", p_value)

# Interpretation based on p-value
if p_value < 0.05:
    print("The tutoring program has a significant effect on exam scores.")
else:
    print("No significant effect of the tutoring program on exam scores.")
```

```
Z-Statistic: 4.016632088371217
P-Value: 5.903578049588008e-05
The tutoring program has a significant effect on exam scores.
```

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. Use the below code to generate samples of respective arrays of blood pressure: "-python before_drug = 150, 140, 135, 155, 160, 152, 148, 130, 1381) after_drug - 140, 132128, 145, 148, 138, 136, 125, 1301) Implement z-test to find if the drug really works or not.

[17]:
```
import numpy as np

# Blood pressure measurements before and after the drug administration
before_drug = np.array([150, 140, 135, 155, 160, 152, 148, 130, 138])
after_drug = np.array([140, 132, 128, 145, 148, 138, 136, 125, 130])
from scipy import stats

# Calculate the differences
differences = before_drug - after_drug
mean_diff = np.mean(differences)
std_diff = np.std(differences, ddof=1)

# Calculate the z-statistic
n = len(differences)
z_stat = mean_diff / (std_diff / np.sqrt(n))

# Calculate the p-value for a one-tailed test (since we're only checking for␣
 ↪reduction)
p_value = stats.norm.sf(abs(z_stat))
```

```
print("Z-Statistic:", z_stat)
print("P-Value:", p_value)

# Interpretation based on p-value
if p_value < 0.05:
    print("The drug has a significant effect in reducing blood pressure.")
else:
    print("The drug does not have a statistically significant effect in␣
 ↪reducing blood pressure.")
```

```
Z-Statistic: 10.117647058823529
P-Value: 2.3068675895993925e-24
The drug has a significant effect in reducing blood pressure.
```

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: response _ times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.41) Implement z-test to find the claims made by customer service department are tru or false.

[18]:
```
# Response times in minutes
response_times = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])
# Hypothesized mean response time
hypothesized_mean = 5

# Calculate the sample mean and standard deviation
sample_mean = np.mean(response_times)
sample_std = np.std(response_times, ddof=1)
n = len(response_times)

# Calculate the z-statistic
z_stat = (sample_mean - hypothesized_mean) / (sample_std / np.sqrt(n))

# Calculate the p-value for a one-tailed test (checking if mean is less than 5)
p_value = stats.norm.cdf(z_stat)

print("Z-Statistic:", z_stat)
print("P-Value:", p_value)

# Interpretation based on p-value
if p_value < 0.05:
    print("The claim that the response time is less than 5 minutes is supported.
 ↪")
else:
    print("The data does not support the claim that the response time is less␣
 ↪than 5 minutes.")
```

```
Z-Statistic: -3.184457226042963
```

```
P-Value: 0.0007251287113068958
The claim that the response time is less than 5 minutes is supported.
```

15

IS. 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 AIB test analysis, including calculating the t-statistic, degrees of freedom, and p—value. Use the following data: ”*python = [28, 32, 33, 29, 31, 34, 30, 35, 36, 371 = [40, 41, 38, 42, 39, 44, 43, 41, 45, 47]

[19]:
```python
import numpy as np
from scipy import stats

# Click-through rates for each layout
layout_A = np.array([28, 32, 33, 29, 31, 34, 30, 35, 36, 37])
layout_B = np.array([40, 41, 38, 42, 39, 44, 43, 41, 45, 47])
def ab_test_analysis(sample1, sample2):
    # Calculate means and standard deviations
    mean1, mean2 = np.mean(sample1), np.mean(sample2)
    std1, std2 = np.std(sample1, ddof=1), np.std(sample2, ddof=1)
    n1, n2 = len(sample1), len(sample2)

    # Calculate the t-statistic
    t_stat, p_value = stats.ttest_ind(sample1, sample2, equal_var=False)

    # Degrees of freedom for Welch's t-test
    df = ((std1**2 / n1 + std2**2 / n2)**2) / (((std1**2 / n1)**2 / (n1 - 1)) +␣
 ↪((std2**2 / n2)**2 / (n2 - 1)))

    print("T-Statistic:", t_stat)
    print("Degrees of Freedom:", df)
    print("P-Value:", p_value)

    if p_value < 0.05:
        print("There is a significant difference in click-through rates between␣
 ↪the two layouts.")
    else:
        print("No significant difference in click-through rates between the two␣
 ↪layouts.")

# Run the A/B test analysis
ab_test_analysis(layout_A, layout_B)
```

```
T-Statistic: -7.298102156175069
Degrees of Freedom: 17.879871863320876
P-Value: 9.19659607078939e-07
There is a significant difference in click-through rates between the two
layouts.
```

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. Use the following data of cholestrol level: [180, 182, 175, 185, 178, 176, 172, 184, 179, 183] = [170, 172, 165, 168, 175, 173, 170, 178, 172, 1761]

[20]:
```
# Cholesterol levels for each drug
existing_drug = np.array([180, 182, 175, 185, 178, 176, 172, 184, 179, 183])
new_drug = np.array([170, 172, 165, 168, 175, 173, 170, 178, 172, 176])
# Run the analysis for the clinical trial data
ab_test_analysis(existing_drug, new_drug)
```

```
T-Statistic: 4.140480986208659
Degrees of Freedom: 17.866770765582338
P-Value: 0.0006229228945469329
There is a significant difference in click-through rates between the two
layouts.
```

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. Use the following data of test score: = [80, 85, 90, 75, 88, 82 92, 78, 85, 871 = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]

[21]:
```
import numpy as np
from scipy import stats

# Pre- and post-intervention math scores
pre_intervention_scores = np.array([80, 85, 90, 75, 88, 82, 92, 78, 85, 87])
post_intervention_scores = np.array([90, 92, 88, 92, 95, 91, 96, 93, 89, 93])
def analyze_intervention(pre_scores, post_scores):
    # Perform paired t-test
    t_stat, p_value = stats.ttest_rel(post_scores, pre_scores)

    print("T-Statistic:", t_stat)
    print("P-Value:", p_value)

    if p_value < 0.05:
        print("The intervention had a significant impact on math scores.")
    else:
        print("The intervention did not have a significant impact on math␣
 ↪scores.")

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

```
T-Statistic: 4.42840883965761
P-Value: 0.0016509548165795493
```

```
The intervention had a significant impact on math scores.
```

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. Use the below code to generate synthetic data: # Generate synthetic salary data for male and female employees np.random.seed(o) # For reproducibility male _ salaries scale—10000, size—20) female _ salaries = np.random.normaI(Ioc=55000, scale=9000, size=20)

[22]:
```
np.random.seed(0)  # For reproducibility
male_salaries = np.random.normal(loc=70000, scale=10000, size=20)  # Mean =␣
↪70000, Std = 10000
female_salaries = np.random.normal(loc=65000, scale=9000, size=20)  # Mean =␣
↪65000, Std = 9000
def gender_salary_gap_analysis(male_salaries, female_salaries):
    # Perform two-sample t-test
    t_stat, p_value = stats.ttest_ind(male_salaries, female_salaries,␣
↪equal_var=False)

    print("T-Statistic:", t_stat)
    print("P-Value:", p_value)

    if p_value < 0.05:
        print("There is a significant difference in average salaries between␣
↪male and female employees.")
    else:
        print("No significant difference in average salaries between male and␣
↪female employees.")

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

```
T-Statistic: 3.252394906430084
P-Value: 0.002481181980805451
There is a significant difference in average salaries between male and female
employees.
```

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. Use the following data: versionl_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, 851 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, 821

[23]:
```
import numpy as np
from scipy import stats

# Quality scores for the two versions
```

```
version1_scores = np.array([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 = np.array([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(version1, version2):
    # Perform two-sample t-test
    t_stat, p_value = stats.ttest_ind(version1, version2, equal_var=False)

    print("T-Statistic:", t_stat)
    print("P-Value:", p_value)

    if p_value < 0.05:
        print("There is a significant difference in quality between the two␣
 ↪versions.")
    else:
        print("No significant difference in quality between the two versions.")

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

```
T-Statistic: 11.325830417646698
P-Value: 1.078754084378157e-14
There is a significant difference in quality between the two versions.
```

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

- [4, 5, 41 branch_a_scores — = [3, 3, 31 branch_b_scores

[24]:
```
# Customer satisfaction scores for the two branches
branch_a_scores = np.array([78, 82, 85, 90, 88, 75, 84, 80, 82, 89, 90, 85, 87,␣
 ↪91, 76, 83, 85, 88, 80, 81])
branch_b_scores = np.array([72, 76, 80, 78, 75, 74, 78, 76, 79, 77, 73, 80, 78,␣
 ↪81, 75, 74, 72, 75, 79, 76])
def analyze_customer_satisfaction(branch_a, branch_b):
    # Perform two-sample t-test
    t_stat, p_value = stats.ttest_ind(branch_a, branch_b, equal_var=False)

    print("T-Statistic:", t_stat)
    print("P-Value:", p_value)

    if p_value < 0.05:
        print("There is a significant difference in customer satisfaction␣
 ↪between the two branches.")
    else:
```

```
        print("No significant difference in customer satisfaction between the␣
 ↪two branches.")

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

```
T-Statistic: 6.231878255307426
P-Value: 7.131966974914861e-07
There is a significant difference in customer satisfaction between the two
branches.
```

[ ]:

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: ”*python #Sample data: Product satisfaction levels (rows) vs. Customer regions (columns) data = 30, 40, 201, [30, 40, 30, 501, [20, 30, 40, 3011)

[25]:
```
import numpy as np
from scipy.stats import chi2_contingency

# Sample data: Product satisfaction levels (rows) vs. Customer regions (columns)
data = np.array([[30, 40, 20, 30],   # Satisfied
                 [30, 40, 30, 50],   # Neutral
                 [20, 30, 40, 30]])  # Dissatisfied

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

print("Chi-Square Statistic:", chi2)
print("P-Value:", p)
print("Degrees of Freedom:", dof)
print("Expected Frequencies:\n", expected)

# Interpret the result
if p < 0.05:
    print("There is a significant relationship between product satisfaction␣
 ↪levels and customer regions.")
else:
    print("No significant relationship between product satisfaction levels and␣
 ↪customer regions.")
```

```
Chi-Square Statistic: 14.239267676767675
P-Value: 0.027074665904629258
Degrees of Freedom: 6
```

```
Expected Frequencies:
 [[24.61538462 33.84615385 27.69230769 33.84615385]
 [30.76923077 42.30769231 34.61538462 42.30769231]
 [24.61538462 33.84615385 27.69230769 33.84615385]]
There is a significant relationship between product satisfaction levels and
customer regions.
```

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 = 30, 20], [30, 40, 301, [20, 30, 40]])

[26]:
```python
# Sample data: Job performance levels before (rows) and after (columns) training
data_performance = np.array([[30, 20, 30],   # Before Training (Effective,␣
 ↪Neutral, Ineffective)
                             [30, 40, 30],   # After Training (Effective,␣
 ↪Neutral, Ineffective)
                             [120, 30, 40]])  # Total counts (optional for␣
 ↪clarity)

# Perform the Chi-Square test
chi2_performance, p_performance, dof_performance, expected_performance =␣
 ↪chi2_contingency(data_performance)

print("Chi-Square Statistic (Performance):", chi2_performance)
print("P-Value (Performance):", p_performance)
print("Degrees of Freedom (Performance):", dof_performance)
print("Expected Frequencies (Performance):\n", expected_performance)

# Interpret the result
if p_performance < 0.05:
    print("There is a significant difference in job performance levels before␣
 ↪and after the training.")
else:
    print("No significant difference in job performance levels before and after␣
 ↪the training.")
```

```
Chi-Square Statistic (Performance): 39.30438596491228
P-Value (Performance): 6.027361018839104e-08
Degrees of Freedom (Performance): 4
Expected Frequencies (Performance):
 [[38.91891892 19.45945946 21.62162162]
 [48.64864865 24.32432432 27.02702703]
 [92.43243243 46.21621622 51.35135135]]
There is a significant difference in job performance levels before and after the
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: # Sample data: Customer satisfaction scores for each product version standard_scores = [80, 85, 90, 78, 88, 82, 92, 78, 85, 871 premium_scores = [90, 92; 88, 92, 95, 91, 96, 93, 89, 931 deluxe_scores = [95, 98, 92, 97, 96, 94, 98, 97, 92, 99]

[27]:
```
import numpy as np
from scipy.stats import f_oneway

# 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]

# Perform the ANOVA test
f_statistic, p_value = f_oneway(standard_scores, premium_scores, deluxe_scores)

# Output the results
print("F-Statistic:", f_statistic)
print("P-Value:", p_value)

# Interpret the results
alpha = 0.05  # Significance level
if p_value < alpha:
    print("There is a significant difference in customer satisfaction scores␣
 ↪among the product versions.")
else:
    print("No significant difference in customer satisfaction scores among the␣
 ↪product versions.")
```

```
F-Statistic: 27.03556231003039
P-Value: 3.5786328857349003e-07
There is a significant difference in customer satisfaction scores among the
product versions.
```

[ ]: