In [21]:
```python
import random
import statistics
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
#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:
```

In [3]:
```python
#(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.
```

In [8]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Mean: ", statistics.mean(int_list))
print("Median:" , statistics.median(int_list))
```

```
Mean:  109.84
Median: 110.0
```

In [5]:
```python
#(ii) Develop a program to compute the mode of a list of integers.
```

In [9]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Mode: ", statistics.mode(int_list))
```

```
Mode:  115
```

In [ ]:
```python
#(iii) Implement a function to calculate the weighted mean of a list of values and their corresponding weights
```

In [12]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
weights = np.random.dirichlet(np.ones(len(int_list)),size = 1)[0]
print("Weighted Mean: ",np.average(int_list,weights =weights))
```

```
Weighted Mean:  113.03992468026703
```

In [ ]:
```python
#(iv) Write a Python function to find the geometric mean of a list of positive numbers.
```

In [14]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Geometric Mean:" , stats.gmean(int_list))
```

```
Geometric Mean: 109.91660612028284
```

In [ ]:
```python
#(v) Create a program to calculate the harmonic mean of a list of values.
```

In [16]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Harmonic Mean:" , statistics.harmonic_mean(int_list))
```

```
Harmonic Mean: 110.45933268218363
```

In [ ]:
```python
#(vi) Build a function to determine the midrange of a list of numbers (average of the minimum and maximum).
```

In [17]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Mid range:" , (min(int_list)+max(int_list))/2)
```

```
Mid range: 110.0
```

In [ ]:
```python
#(vii) Implement a Python program to find the trimmed mean of a list, excluding a certain percentage of outliers.
```

In [20]:
```python
int_list = [random.randint(90,130)for _ in range(100)]
print("Trimmed mean(10%):" ,np.mean(np.sort(int_list)[10:-10]))
```

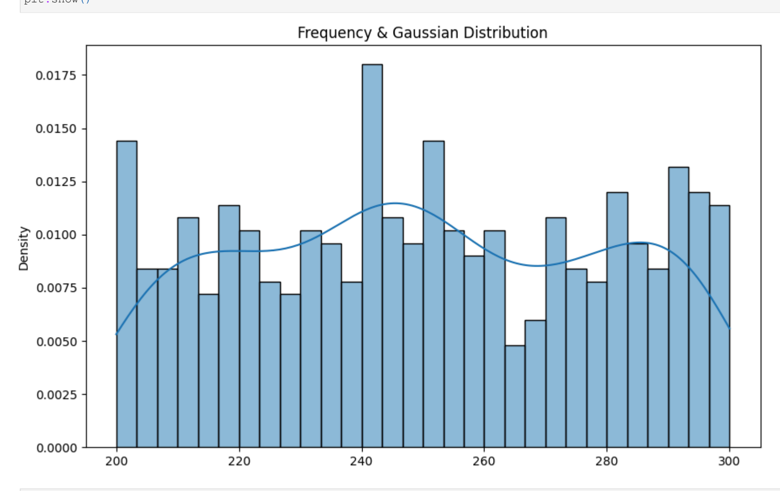
```
Trimmed mean(10%): 110.0625
```

In [ ]:
```python
#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:
```

In [ ]:
```python
#(i) Compare the given list of visualization for the given data:
```

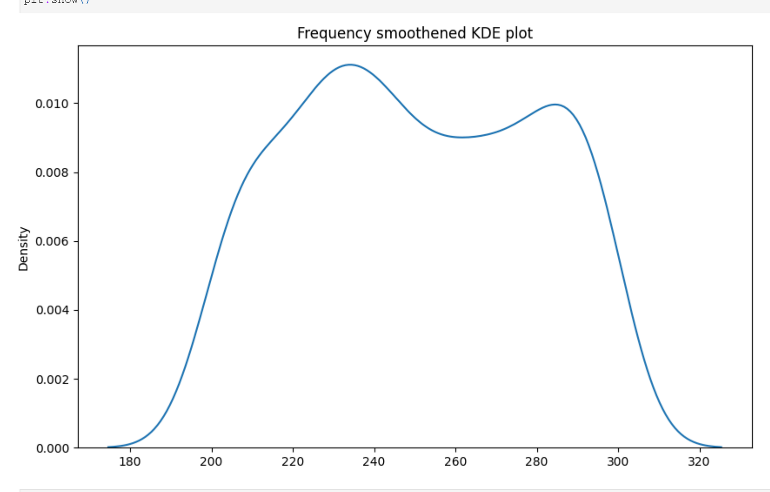
In [ ]:
```python
#1. Frequency & Gaussian distribution
```

In [24]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

plt.figure(figsize =(10,6))
sns.histplot(int_list2, bins = 30,kde = True , stat = "density")
plt.title("Frequency & Gaussian Distribution")
plt.show()
```



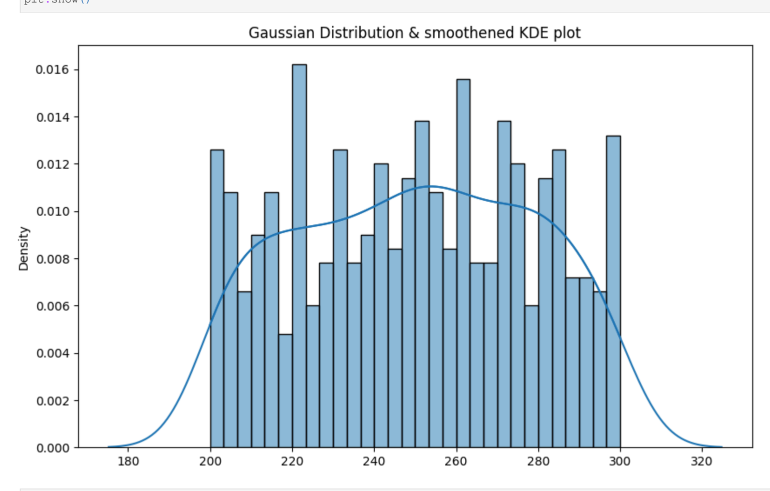
In [ ]:
```python
#2.Frequency smoothened KDE plot
```

In [27]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

plt.figure(figsize =(10,6))
sns.kdeplot(int_list2)
plt.title("Frequency smoothened KDE plot")
plt.show()
```



In [ ]:
```python
#3.Gaussian distribution & smoothened KDE plot
```

In [28]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

plt.figure(figsize =(10,6))
sns.histplot(int_list2, bins = 30,kde = True , stat = "density")
sns.kdeplot(int_list2)
plt.title("Gaussian Distribution & smoothened KDE plot")
plt.show()
```



In [ ]:
```python
#(ii) Write a Python function to calculate the range of a given list of numbers.
```

In [29]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

print("Range: " , max(int_list2) - min(int_list2))
```

```
Range:  100
```

In [ ]:
```python
#(iii) Create a program to find the variance and standard deviation of a list of numbers.
```

In [30]:
```python
import random
import math

int_list2 = [random.randint(200,300) for _ in range(500)]
print("Variance: " , statistics.variance(int_list2))
print("Standatd deviation : " , statistics.stdev(int_list2))
```

```
Variance:  905.3535711422845
Standatd deviation :  30.089093890349783
```

In [ ]:
```python
#(iv) Implement a function to compute the interquartile range (IQR) of a list of values.
```

In [31]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

q75 , q25 = np.percentile(int_list2, [75 ,25])
iqr = q75 - q25

print("Interquartile Range(IQR):" , iqr)
```

```
Interquartile Range(IQR): 50.5
```

In [ ]:
```python
#(v) Build a program to calculate the coefficient of variation for a dataset.
```

In [32]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

cv = np.std(int_list2) / np.mean(int_list2)* 100
print("coefficient of variation(cv): " , cv)
```

```
coefficient of variation(cv):  11.892439901885568
```

In [ ]:
```python
#(vi) Write a Python function to find the mean absolute deviation (MAD) of a list of numbers.
```

In [33]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

mad = np.mean([abs(x - np.mean(int_list2)) for x in int_list2])
print("Mean Absolute Deviaiton(MAD):" , mad)
```

```
Mean Absolute Deviaiton(MAD): 26.0544
```

In [ ]:
```python
#(vii) Create a program to calculate the quartile deviation of a list of values.
```

In [35]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]

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

```
Quartile Deviation :  26.0
```

In [ ]:
```python
#(viii) Implement a function to find the range-based coefficient of dispersion for a dataset.
```

In [41]:
```python
int_list2 = [random.randint(200,300) for _ in range(500)]
dis_coff = ((max(int_list2) - min(int_list2))/ (max(int_list2) +min(int_list2)))*100
print("Range-Based coffiient of Disperson: " , dis_coff)
```

```
Range-Based coffiient of Disperson:  20.0
```

In [ ]:
```python
#3.Write a Python class representing a discrete random variable with methods to calculate its expected value and variance.
```

In [44]:
```python
v , p =  [1,2,3,4,5] , [0.1 , 0.2 , 0.3 , 0.2 , 0.2]
print("Expected value : " , np.average(v,weights = p))
print("Variance: " , np.var(v, ddof = 1))
```

```
Expected value :  3.2
Variance:  2.5
```

In [ ]:
```python
#4.Implement a program to simulate the rolling of a fair six-sided die and calculate the expected value and variance of the outcomes.
```

In [46]:
```python
o =[random.randint(1,6) for _ in range(10000)]
print("Expected value: " , sum(o)/len(o))
print("Variance : " , sum((x - sum(o)/len(o))**2 for x in o)/len(o))
```

```
Expected value:  3.4997
Variance :  2.90819991
```

In [ ]:
```python
#5.Create a Python function to generate random samples from a given probability distribution (e.g., binomial, Poisson) and calculate their mean and variance.
```

In [53]:
```python
dist , params , size = 'binomial' , [10,0.5],1000
s = np.random.choice([np.random.binomial, np.random.poisson])(*params , size)
print(f"Mean: {s.mean()} , variance:{s.var()}")
```

```
Mean: 5.026 , variance:2.6193239999999998
```

In [ ]:
```python
#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.
```

In [54]:
```python
mean, std_dev, size = 0, 1, 1000
samples = np.random.normal(mean, std_dev, size)
print(f"Mean: {np.mean(samples)}")
print(f"Variance: {np.var(samples)}")
print(f"Standard Deviation: {np.std(samples)}")
```

```
Mean: -0.04448485817560295
Variance: 1.0216830176845484
Standard Deviation: 1.0107833683260465
```

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

In [ ]:
```python
#(i) Write a Python function that calculates their skewness.
```

In [58]:
```python
import seaborn as sns
from scipy.stats import skew

tips = sns.load_dataset('tips')
print(skew(tips['total_bill']))
print(skew(tips['tip']))
```

```
1.1262346334818638
1.4564266884221506
```

In [ ]:
```python
#(ii) Create a program that determines whether the columns exhibit positive skewness, negative skewness, or is approximately symmetric.
```

In [59]:
```python
tips = sns.load_dataset('tips')
for col in ['total_bill', 'tip']:
    print(f"{col.capitalize()}: {skew(tips[col])} ({'+' if skew(tips[col]) > 0 else '-' if skew(tips[col]) < 0 else 'Symmetric'} skewness)")
```

```
Total_bill: 1.1262346334818638 (+ skewness)
Tip: 1.4564266884221506 (+ skewness)
```

In [ ]:
```python
#(iii) Write a function that calculates the covariance between two columns.
```

In [60]:
```python
tips = sns.load_dataset('tips')
print(tips['total_bill'].cov(tips['tip']))
```

```
8.323501629224854
```

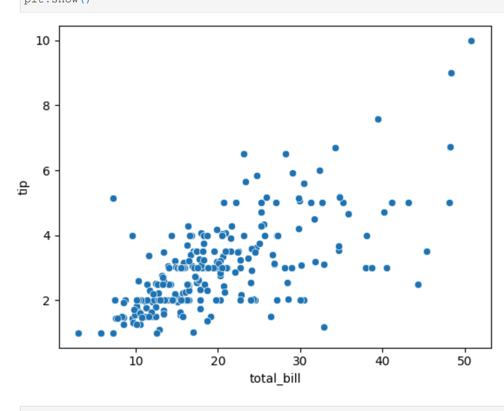
In [ ]:
```python
#(iv) Implement a Python program that calculates the Pearson correlation coefficient between two columns.
```

In [61]:
```python
correlation = tips['total_bill'].corr(tips['tip'])

print(f"Pearson Correlation Coefficient: {correlation}")
```

```
Pearson Correlation Coefficient: 0.6757341092113645
```

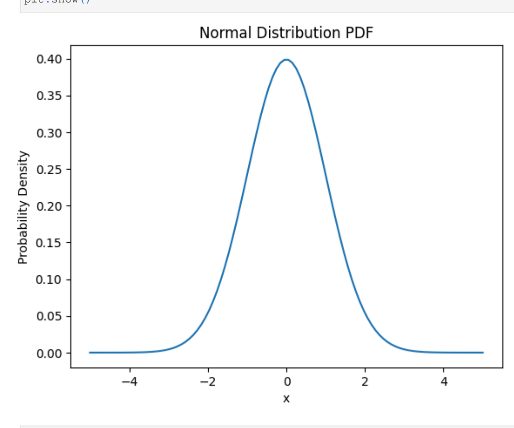
In [ ]:
```python
#(v) Write a script to visualize the correlation between two specific columns in a Pandas DataFrame using scatter plots.
```

In [62]:
```python
tips = sns.load_dataset('tips')
sns.scatterplot(x='total_bill', y='tip', data=tips)
plt.show()
```



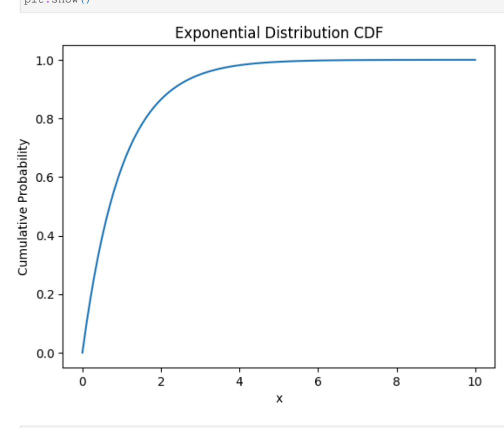
In [ ]:
```python
#8.Write a Python function to calculate the probability density function (PDF) of a continuous random variable for a given normal distribution.
```

In [63]:
```python
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

x = np.linspace(-5, 5, 100)
plt.plot(x, norm.pdf(x))
plt.xlabel('x'); plt.ylabel('Probability Density'); plt.title('Normal Distribution PDF')
plt.show()
```



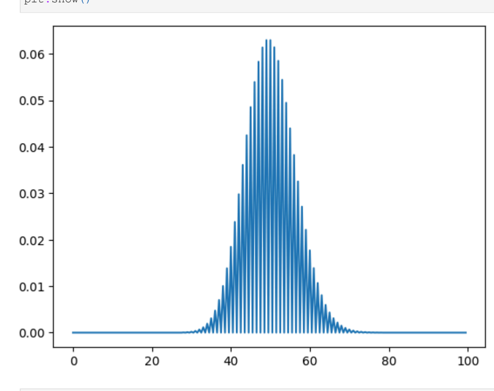
In [ ]:
```python
#9.Create a program to calculate the cumulative distribution function (CDF) of exponential distribution.
```

In [64]:
```python
lam = 1
x = np.linspace(0, 10, 100)
cdf = 1 - np.exp(-lam * x)

plt.plot(x, cdf)
plt.xlabel('x')
plt.ylabel('Cumulative Probability')
plt.title('Exponential Distribution CDF')
plt.show()
```



In [ ]:
```python
#10.Write a Python function to calculate the probability mass function (PMF) of Poisson distribution.
```

In [89]:
```python
from scipy.stats import poisson
x = np.arange(0, 100, 0.5)
y = poisson.pmf(x, mu=40, loc=10)
plt.plot(x, y)
plt.show()
```



In [ ]:
```python
#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.
```

In [106…
```python
# 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.

import numpy as np
import statsmodels.stats.proportion as proportion

old = np.array([1] * 50 + [0] * 950)
new = np.array([1] * 70 + [0] * 930)

z, p = proportion.proportions_ztest([sum(new), sum(old)], [len(new), len(old)], alternative='larger')

if p < 0.05:
    print("New layout is better.")
else:
    print("No significant difference.")
```

```
New layout is better.
```

In [ ]:
```python
#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.
```

In [103…
```python
# 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.


before_program , after_program = np.array([75, 80, 85, 70, 90, 78, 92, 88, 82, 87]), np.array([80, 85, 90, 80, 92, 80, 95, 90, 85, 88])
z = (np.mean(after_program) - np.mean(before_program )) / (np.std(before_program , ddof=1) / np.sqrt(len(before_program )))
p_val = 2 * norm.sf(abs(z))

print(f"Z-score: {z}, P-value: {p_val}")
print("Tutor's claims are true" if p_val < 0.05 else "Tutor's claims are false")
```

```
Z-score: 1.714528046981441, P-value: 0.0864317906048479
Tutor's claims are false
```

In [ ]:
```python
#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.
```

In [73]:
```python
# 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.

b, a = np.array([145, 150, 140, 135, 155, 160, 152, 148, 130, 138]), np.array([130, 140, 132, 128, 145, 148, 138, 136, 125, 130])
z = (np.mean(b) - np.mean(a)) / (np.std(b, ddof=1) / np.sqrt(len(b)))
p_val = 2 * norm.sf(abs(z))

print(f"Z-score: {z}, P-value: {p_val}")
print("Drug is effective" if p_val < 0.05 else "Drug is not effective")
```

```
Z-score: 3.374800990700482, P-value: 0.0007386908781037341
Drug is effective
```

In [ ]:
```python
#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.
```

In [74]:
```python
# 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.

r = np.array([4.3, 3.8, 5.1, 4.9, 4.7, 4.2, 5.2, 4.5, 4.6, 4.4])
z = (np.mean(r) - 5) / (np.std(r, ddof=1) / np.sqrt(len(r)))
p_val = norm.sf(abs(z))

print(f"Z-score: {z}, P-value: {p_val}")
print("Claims are false" if p_val < 0.05 else "Claims are true")
```

```
Z-score: -3.184457226042963, P-value: 0.0007251287113068958
Claims are false
```

In [ ]:

In [ ]:
```python
#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,
```

In [ ]:

In [78]:
```python
#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]

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

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


ma, mb = np.mean(a), np.mean(b)
sa, sb = np.std(a, ddof=1), np.std(b, ddof=1)
n = len(a)

t = (mb - ma) / np.sqrt((sa**2 / n) + (sb**2 / n))
df = 2 * n - 2

print(f"T-statistic: {t}, Degrees of freedom: {df}")
```

```
T-statistic: 7.298102156175071, Degrees of freedom: 18
```

In [ ]:

In [ ]:
```python
#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
```

In [79]:
```python
# 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]

ma, mb = np.mean(existing_drug_levels), np.mean(new_drug_levels)
sa, sb = np.std(existing_drug_levels, ddof=1), np.std(new_drug_levels, ddof=1)
n = len(existing_drug_levels)

t = (mb - ma) / np.sqrt((sa**2 / n) + (sb**2 / n))
df = 2 * n - 2

print(f"T-statistic: {t}")
print(f"Degrees of freedom: {df}")
```

```
T-statistic: -4.140480986208661
Degrees of freedom: 18
```

In [ ]:
```python
#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-va
```

In [81]:
```python
# 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]

# import numpy as np
from scipy.stats import ttest_rel

a = [80, 85, 90, 75, 88, 82, 92, 78, 85, 87]
b = [90, 92, 88, 92, 95, 91, 96, 93, 89, 93]
t, p = ttest_rel(a, b)

print(f"T-statistic: {t}, P-value: {p}")
print("Intervention was effective" if p < 0.05 else "Intervention was not effective")
```

```
T-statistic: -4.42840883965761, P-value: 0.0016509548165795493
Intervention was effective
```

In [ ]:
```python
#18.An HR department wants to investigate if there's a gender-based salary gap within the company. Developa program to analyze salary data, calculate the t-statistic, and determine if there's a statistica
```

In [101…
```python
# Use the below code to generate synthetic data:


# ```python

# Generate synthetic salary data for male and female employees

np.random.seed(0)  # For reproducibility


import numpy as np
from scipy import stats

np.random.seed(0)
male_salaries = np.random.normal(50000, 10000, 20)
female_salaries = np.random.normal(55000, 9000, 20)

t, p = stats.ttest_ind(male_salaries, female_salaries , equal_var=False)

if p < 0.05:
    print("Salary gap exists.")
else:
    print("No significant gap.")
```

```
No significant gap.
```

In [ ]:
```python
#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 decid
```

In [84]:
```python
# 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]



m1, m2 = np.mean(version1_scores), np.mean(version2_scores)
s1, s2 = np.std(version1_scores, ddof=1), np.std(version2_scores, ddof=1)
n1, n2 = len(version1_scores), len(version2_scores)

t = (m2 - m1) / np.sqrt((s1**2 / n1) + (s2**2 / n2))
df = n1 + n2 - 2

print(f"T-statistic: {t}")
print(f"Degrees of freedom: {df}")
```

```
T-statistic: -11.325830417646698
Degrees of freedom: 48
```

In [ ]:
```python
#20.A restaurant chain collects customer satisfaction scores for two different branches. Write a program toanalyze the scores, calculate the t-statistic, and determine if there's a statistically significa
```

In [91]:
```python
# 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]


t_stat = (np.mean(branch_b_scores) - np.mean(branch_a_scores)) / np.sqrt((np.std(branch_a_scores, ddof=1)**2 / len(branch_a_scores)) + (np.std(branch_b_scores, ddof=1)**2 / len(branch_b_scores)))
print(f"T-statistic: {t_stat}")
```

```
T-statistic: -5.480077554195743
```

In [ ]:
```python
#21.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 cl
```

In [94]:
```python
# Use the below code to generate data:

# ```python

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)

observed = np.array([[sum((age_groups == '18-30') & (voter_preferences == 'Candidate A')), sum((age_groups == '18-30') & (voter_preferences == 'Candidate B'))],
                     [sum((age_groups == '31-50') & (voter_preferences == 'Candidate A')), sum((age_groups == '31-50') & (voter_preferences == 'Candidate B'))],
                     [sum((age_groups == '51+') & (voter_preferences == 'Candidate A')), sum((age_groups == '51+') & (voter_preferences == 'Candidate B'))]])

expected = np.outer(observed.sum(axis=1), observed.sum(axis=0)) / observed.sum()

chi2 = ((observed - expected) ** 2 / expected).sum()
print("Chi-Square statistic: " + str(chi2))
```

```
Chi-Square statistic: 1.4401669758812612
```

In [ ]:
```python
#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 cust
```

In [100…
```python
# 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]])


import numpy as np
from scipy.stats import chi2_contingency

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

chi2, p, _, _ = chi2_contingency(data)

if p < 0.05:
    print("Satisfaction and region are related.")
else:
    print("Satisfaction and region are independent.")
```

```
Satisfaction and region are related.
```

In [ ]:
```python
#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
```

In [99]:
```python
# 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]])
import numpy as np
from scipy.stats import chi2_contingency

data = np.array([[50, 30, 20], [30, 40, 30], [20, 30, 40]])
chi2, p, _, _ = chi2_contingency(data)

if p < 0.05:
    print("Training changed job performance.")
else:
    print("No significant change.")
```

```
Training changed job performance.
```

In [ ]:
```python
#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 t
```

In [97]:
```python
# 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]


f_statistic, p_value = stats.f_oneway(standard_scores, premium_scores, deluxe_scores)

print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")


alpha = 0.05

if p_value < alpha:
    print("There is a significant difference in customer satisfaction scores among the three product versions.")
else:
    print("There is no significant difference in customer satisfaction scores among the three product versions.")
```

```
F-statistic: 27.0356
P-value: 0.0000
There is a significant difference in customer satisfaction scores among the three product versions.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: