Part B - Lab Questions

In [119…

```
"""Q1.(a) Create a 1D NumPy array with values 1 to 20.
(b) Extract all prime numbers from it.
(c) Compute the mean and variance of the extracted primes."""

# (a) Create a 1D NumPy array with values 1 to 20
import numpy as np
ar = np.arange(1,21)
print("Array is:",ar)

# (b) Extract all prime numbers
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(np.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True

primes = np.array([x for x in ar if is_prime(x)])
print("Prime numbers:", primes)

# (c) Compute mean and variance of primes
mean_primes = np.mean(primes)
var_primes = np.var(primes)
print("Mean of primes:", mean_primes)
print("Variance of primes:", var_primes)
```

```
Array is: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
Prime numbers: [ 2  3  5  7 11 13 17 19]
Mean of primes: 9.625
Variance of primes: 35.734375
```

In [120…

```
"""Q2.(a) Create a 4×4 NumPy array with numbers 1 to 16.
(b) Extract the 2×2 bottom-left sub-matrix.
(c) Compute the determinant of the sub-matrix."""

import numpy as np
# (a) Create a 4×4 NumPy array with numbers 1 to 16
arr = np.arange(1, 17).reshape(4, 4)
print("4x4 Array:\n", arr)

# (b) Extract the 2×2 bottom-left sub-matrix
sub_matrix = arr[2:, :2]
print("2x2 Bottom-left Sub-matrix:\n", sub_matrix)

# (c) Compute the determinant of the sub-matrix
det = np.linalg.det(sub_matrix)
print("Determinant of sub-matrix:", det)
```

```
4x4 Array:
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
2x2 Bottom-left Sub-matrix:
 [[ 9 10]
 [13 14]]
Determinant of sub-matrix: -4.000000000000009
```

In [121…

```python
"""Q3.(a) Create a DataFrame with 5 students and marks in 3 subjects.
(b) Add a column for total and average marks.
(c) Identify the topper and print their name with average."""

import pandas as pd
#(a) Create a DataFrame with 5 students and marks in 3 subjects
data = {
    'Name': ['Arpit', 'Mohan', 'Gagan', 'Chetan', 'Isha'],
    'Maths': [78, 85, 92, 88, 76],
    'Science': [82, 79, 95, 91, 80],
    'English': [74, 88, 90, 85, 78]
}
df = pd.DataFrame(data)
print("Student Marks DataFrame:\n", df)

#(b) Add total and average columns
df['Total'] = df[['Maths', 'Science', 'English']].sum(axis=1)
df['Average'] = df['Total'] / 3
print("\nDataFrame with Total & Average:\n", df)

#(c) Identify the topper and print their name with average
topper = df.loc[df['Average'].idxmax()]
print(f"\nTopper: {topper['Name']} | Average Marks: {topper['Average']:.2f}")
```

```
Student Marks DataFrame:
      Name  Maths  Science  English
0   Arpit     78       82       74
1   Mohan     85       79       88
2   Gagan     92       95       90
3  Chetan     88       91       85
4    Isha     76       80       78

DataFrame with Total & Average:
      Name  Maths  Science  English  Total    Average
0   Arpit     78       82       74    234  78.000000
1   Mohan     85       79       88    252  84.000000
2   Gagan     92       95       90    277  92.333333
3  Chetan     88       91       85    264  88.000000
4    Isha     76       80       78    234  78.000000

Topper: Gagan | Average Marks: 92.33
```

In [122…

```python
"""Q4.(a) Simulate 1000 coin tosses using NumPy (1=Head, 0=Tail).
(b) Count frequency of heads and tails.
```

```
(c) Estimate probability of heads. Is it close to 0.5? Why/Why not?"""

import numpy as np
#(a) Simulate 1000 coin tosses (1=Head, 0=Tail)
tosses = np.random.randint(0, 2, 1000)
print("Coin Toss Results (first 25 shown):", tosses[:25])

#(b) Count frequency of heads and tails
heads = np.count_nonzero(tosses == 1)
tails = np.count_nonzero(tosses == 0)
print(f"Heads: {heads}, Tails: {tails}")

#(c) Estimate probability of heads
prob_heads = heads / len(tosses)
print(f"Probability of Heads: {prob_heads:.3f}")

#Commenting on result
if abs(prob_heads - 0.5) < 0.05:
    print("Yes, it's close to 0.5 — randomness averages out over many tosses."
else:
    print("Not very close to 0.5 — randomness can still cause small deviations
```

```
Coin Toss Results (first 25 shown): [1 0 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 1
0 0 0 0]
Heads: 505, Tails: 495
Probability of Heads: 0.505
Yes, it's close to 0.5 — randomness averages out over many tosses.
```

In [123…

```
"""Q5.(a) Create a DataFrame of employees with columns: ID, Name, Salary.
(b) Add a Bonus column = 10% of Salary.
(c) Display employees with salary above average."""

#(a) Create a DataFrame of employees with columns: ID, Name, Salary.
import pandas as pd
data = {
    'ID': [101, 102, 103, 104, 105],
    'Name': ['Arpit', 'Gagan', 'Mohan', 'Chetan', 'Isha'],
    'Salary': [65000, 52000, 48000, 60000, 55000]
}
df = pd.DataFrame(data)
print("Employees DataFrame:\n",df)

# (b) Add Bonus column = 10% of Salary
df['Bonus'] = df['Salary'] * 0.10
print("\nDataFrame with Bonus:\n", df)

# (c) Display employees with salary above average
avg_salary = df['Salary'].mean()
above_avg = df[df['Salary'] > avg_salary]
print(f"\nAverage Salary: {avg_salary:.2f}")
print("\nEmployees with Salary above Average:\n", above_avg)
```

```
Employees DataFrame:
     ID    Name  Salary
0  101   Arpit   65000
1  102   Gagan   52000
2  103   Mohan   48000
3  104  Chetan   60000
4  105    Isha   55000

DataFrame with Bonus:
     ID    Name  Salary   Bonus
0  101   Arpit   65000  6500.0
1  102   Gagan   52000  5200.0
2  103   Mohan   48000  4800.0
3  104  Chetan   60000  6000.0
4  105    Isha   55000  5500.0

Average Salary: 56000.00

Employees with Salary above Average:
     ID    Name  Salary   Bonus
0  101   Arpit   65000  6500.0
3  104  Chetan   60000  6000.0
```

In [124…

```python
"""Q6.(a) Create a 3×3 NumPy array with values 1 to 9.
(b) Find its transpose and inverse.
(c) Verify that A × A⁻¹ ≈ I."""

import numpy as np

# (a) Create a 3×3 NumPy array with values 1 to 9
A = np.arange(1, 10).reshape(3, 3)
print("Matrix A:\n", A)

# (b) Find its transpose and inverse
A_T = A.T
try:
    A_inv = np.linalg.inv(A)
    print("\nTranspose of A:\n", A_T)
    print("\nInverse of A:\n", A_inv)
except np.linalg.LinAlgError:
    print("\nMatrix A is singular and cannot be inverted.")

# (c) Verify that A × A⁻¹ ≈ I
if 'A_inv' in locals():
    I_approx = np.dot(A, A_inv)
    print("\nA × A⁻¹ ≈ I:\n", I_approx)
```

```
Matrix A:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix A is singular and cannot be inverted.
```

In [125…

```
"""Q7.(a) Generate random daily temperatures (30 values, range 20–40°C).
(b) Find the hottest and coldest day.
(c) Compute mean, median, and standard deviation of temperatures."""

import numpy as np

# (a) Generate random daily temperatures (30 values, range 20–40°C)
temps = np.random.uniform(20, 40, 30)
print("Daily Temperatures (°C):\n", np.round(temps, 2))

# (b) Find the hottest and coldest day
hottest = np.max(temps)
coldest = np.min(temps)
print(f"\nHottest Temperature: {hottest:.2f}°C")
print(f"Coldest Temperature: {coldest:.2f}°C")

# (c) Compute mean, median, and standard deviation
mean_temp = np.mean(temps)
median_temp = np.median(temps)
std_temp = np.std(temps)
print(f"\nMean: {mean_temp:.2f}°C")
print(f"Median: {median_temp:.2f}°C")
print(f"Standard Deviation: {std_temp:.2f}")
```

```
Daily Temperatures (°C):
 [38.   28.09 37.91 21.3  33.71 24.62 30.78 35.46 34.93 34.43 29.66 23.88
 24.11 30.71 28.76 39.5  26.02 21.58 34.54 39.51 35.64 24.94 20.73 28.14
 37.56 28.64 25.68 38.71 35.45 32.22]

Hottest Temperature: 39.51°C
Coldest Temperature: 20.73°C

Mean: 30.84°C
Median: 30.74°C
Standard Deviation: 5.77
```

In [126…

```
"""Q8.(a) Create a Pandas Series of marks for 8 students.
(b) Replace all marks below 40 with “Fail”.
(c) Count how many students passed."""

import pandas as pd
# (a) Create a Pandas Series of marks for 8 students
marks = pd.Series([55, 38, 72, 91, 47, 30, 65, 84])
print("Marks Series:\n", marks)

# (b) Replace all marks below 40 with “Fail”
marks_updated = marks.apply(lambda x: "Fail" if x < 40 else x)
print("\nUpdated Marks (with 'Fail'):\n", marks_updated)

# (c) Count how many students passed
passed_count = sum(marks_updated != "Fail")
print(f"\nNumber of Students Passed: {passed_count}")
```

```
Marks Series:
 0    55
1    38
2    72
3    91
4    47
5    30
6    65
7    84
dtype: int64

Updated Marks (with 'Fail'):
 0      55
1    Fail
2      72
3      91
4      47
5    Fail
6      65
7      84
dtype: object

Number of Students Passed: 6
```

In [127…

```python
"""Q9.(a) Generate 500 random integers between 1 and 6 (simulate dice rolls).
(b) Count how many times each face appears.
(c) Compute relative frequencies and compare with theoretical 1/6."""

import numpy as np

# (a) Generate 500 random integers between 1 and 6 (simulate dice rolls)
rolls = np.random.randint(1, 7, 500)
print("First 20 Dice Rolls:", rolls[:20])

# (b) Count how many times each face appears
counts = {face: np.count_nonzero(rolls == face) for face in range(1, 7)}
print("\nFace Counts:", counts)

# (c) Compute relative frequencies and compare with theoretical 1/6
rel_freq = {face: count / 500 for face, count in counts.items()}
print("\nRelative Frequencies:", rel_freq)
print("\nTheoretical Probability (1/6) ≈ 0.1667")
for face, freq in rel_freq.items():
    print(f"Face {face}: {freq:.3f} (diff = {abs(freq - 1/6):.3f})")
```

```
First 20 Dice Rolls: [3 6 5 4 5 2 1 1 2 1 4 6 3 1 2 3 2 4 2 6]

Face Counts: {1: 93, 2: 74, 3: 81, 4: 81, 5: 86, 6: 85}

Relative Frequencies: {1: 0.186, 2: 0.148, 3: 0.162, 4: 0.162, 5: 0.172, 6: 0.1
7}

Theoretical Probability (1/6) ≈ 0.1667
Face 1: 0.186 (diff = 0.019)
Face 2: 0.148 (diff = 0.019)
Face 3: 0.162 (diff = 0.005)
Face 4: 0.162 (diff = 0.005)
Face 5: 0.172 (diff = 0.005)
Face 6: 0.170 (diff = 0.003)
```

In [128…

```python
"""Q10.(a) Create a DataFrame with 6 products (Name, Quantity, Price).
(b) Add a column “Total = Quantity × Price”.
(c) Find which product generated maximum sales revenue."""

import pandas as pd

# (a) Create a DataFrame with 6 products
data = {
    'Name': ['Pen', 'Notebook', 'Eraser', 'Marker', 'Stapler', 'Folder'],
    'Quantity': [50, 30, 100, 20, 15, 40],
    'Price': [5, 20, 2, 15, 50, 10]
}
df = pd.DataFrame(data)
print("Products DataFrame:\n", df)

# (b) Add a column "Total = Quantity × Price"
df['Total'] = df['Quantity'] * df['Price']
print("\nDataFrame with Total Sales:\n", df)

# (c) Find which product generated maximum sales revenue
max_sales_product = df.loc[df['Total'].idxmax()]
print(f"\nProduct with Maximum Sales: {max_sales_product['Name']} | Revenue: {
```

```
Products DataFrame:
        Name  Quantity  Price
0       Pen        50      5
1  Notebook        30     20
2    Eraser       100      2
3    Marker        20     15
4   Stapler        15     50
5    Folder        40     10

DataFrame with Total Sales:
        Name  Quantity  Price  Total
0       Pen        50      5    250
1  Notebook        30     20    600
2    Eraser       100      2    200
3    Marker        20     15    300
4   Stapler        15     50    750
5    Folder        40     10    400

Product with Maximum Sales: Stapler | Revenue: 750
```

In [129…

```python
"""Q11.(a) Create a DataFrame with some missing values (NaN).
(b) Fill missing values with column mean.
(c) Drop rows where more than 1 value is missing."""

import pandas as pd
import numpy as np

# (a) Create a DataFrame with some missing values
data = {
    'A': [10, 20, np.nan, 40, np.nan],
    'B': [5, np.nan, 15, np.nan, 25],
    'C': [11, np.nan, 12, 16, 20]
}
df = pd.DataFrame(data)
print("Original DataFrame:\n", df)

# (b) Fill missing values with column mean
df_filled = df.fillna(df.mean())
print("\nDataFrame after filling NaNs with column mean:\n", df_filled)

# (c) Drop rows where more than 1 value is missing
df_dropped = df.dropna(thresh=df.shape[1]-1)  # keep rows with at least 2 non-
print("\nDataFrame after dropping rows with >1 missing value:\n", df_dropped)
```

```
Original DataFrame:
       A     B     C
0  10.0   5.0  11.0
1  20.0   NaN   NaN
2   NaN  15.0  12.0
3  40.0   NaN  16.0
4   NaN  25.0  20.0

DataFrame after filling NaNs with column mean:
            A     B      C
0  10.000000   5.0  11.00
1  20.000000  15.0  14.75
2  23.333333  15.0  12.00
3  40.000000  15.0  16.00
4  23.333333  25.0  20.00

DataFrame after dropping rows with >1 missing value:
       A     B     C
0  10.0   5.0  11.0
2   NaN  15.0  12.0
3  40.0   NaN  16.0
4   NaN  25.0  20.0
```

In [130…

```python
"""Q12.(a) Create a NumPy array with integers from 1 to 30.
(b) Reshape it into a 5×6 matrix.
(c) Extract all even numbers from the matrix and compute their average."""

import numpy as np

# (a) Create a NumPy array with integers from 1 to 30
arr = np.arange(1, 31)
print("Array:", arr)

# (b) Reshape into a 5×6 matrix
matrix = arr.reshape(5, 6)
print("\n5x6 Matrix:\n", matrix)

# (c) Extract all even numbers and compute their average
evens = matrix[matrix % 2 == 0]
average_evens = np.mean(evens)
print("\nEven Numbers:", evens)
print("Average of Even Numbers:", average_evens)
```

```
Array: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
 25 26 27 28 29 30]

5x6 Matrix:
 [[ 1  2  3  4  5  6]
 [ 7  8  9 10 11 12]
 [13 14 15 16 17 18]
 [19 20 21 22 23 24]
 [25 26 27 28 29 30]]

Even Numbers: [ 2  4  6  8 10 12 14 16 18 20 22 24 26 28 30]
Average of Even Numbers: 16.0
```

In [131…

```python
"""Q13.
(a) Create a DataFrame of 6 students with columns: Name, Age, Marks.
(b) Select only students who scored above the overall average marks.
(c) Display names of students younger than 20 whose marks are above 60"""

import pandas as pd

# (a) Create a DataFrame of 6 students
data = {
    'Name': ['Arpit', 'Bhaskar', 'Gagan', 'Mohan', 'Isha', 'Farhan'],
    'Age': [19, 21, 18, 22, 20, 17],
    'Marks': [65, 58, 72, 49, 80, 68]
}
df = pd.DataFrame(data)
print("Students DataFrame:\n", df)

# (b) Select students who scored above overall average marks
avg_marks = df['Marks'].mean()
above_avg = df[df['Marks'] > avg_marks]
print(f"\nOverall Average Marks: {avg_marks:.2f}")
print("Students with Marks above Average:\n", above_avg)

# (c) Display names of students younger than 20 whose marks are above 60
young_high = df[(df['Age'] < 20) & (df['Marks'] > 60)]
print("\nStudents younger than 20 with Marks > 60:\n", young_high['Name'].toli
```

```
Students DataFrame:
       Name  Age  Marks
0    Arpit   19     65
1  Bhaskar   21     58
2    Gagan   18     72
3    Mohan   22     49
4     Isha   20     80
5   Farhan   17     68

Overall Average Marks: 65.33
Students with Marks above Average:
      Name  Age  Marks
2   Gagan   18     72
4    Isha   20     80
5  Farhan   17     68

Students younger than 20 with Marks > 60:
 ['Arpit', 'Gagan', 'Farhan']
```

In [132…

```python
"""Q14.(a) Create a Pandas DataFrame with 5 employees having columns: Name, Ta
Hours_Worked.
(b) Add a new column Efficiency = Tasks_Completed / Hours_Worked.
(c) Identify the employee with the highest efficiency and print their name and

import pandas as pd

# (a) Create a DataFrame with 5 employees
data = {
    'Name': ['Arpit', 'Mohan', 'Chetan', 'Bhasker', 'Isha'],
    'Tasks_Completed': [50, 40, 60, 55, 45],
    'Hours_Worked': [10, 8, 15, 12, 9]
}
df = pd.DataFrame(data)
print("Employees DataFrame:\n", df)

# (b) Add a new column Efficiency = Tasks_Completed / Hours_Worked
df['Efficiency'] = df['Tasks_Completed'] / df['Hours_Worked']
print("\nDataFrame with Efficiency:\n", df)

# (c) Identify the employee with the highest efficiency
top_employee = df.loc[df['Efficiency'].idxmax()]
print(f"\nHighest Efficiency: {top_employee['Name']} | Value: {top_employee['E
```

```
Employees DataFrame:
       Name  Tasks_Completed  Hours_Worked
0    Arpit               50            10
1    Mohan               40             8
2   Chetan               60            15
3  Bhasker               55            12
4     Isha               45             9

DataFrame with Efficiency:
       Name  Tasks_Completed  Hours_Worked  Efficiency
0    Arpit               50            10    5.000000
1    Mohan               40             8    5.000000
2   Chetan               60            15    4.000000
3  Bhasker               55            12    4.583333
4     Isha               45             9    5.000000

Highest Efficiency: Arpit | Value: 5.00
```

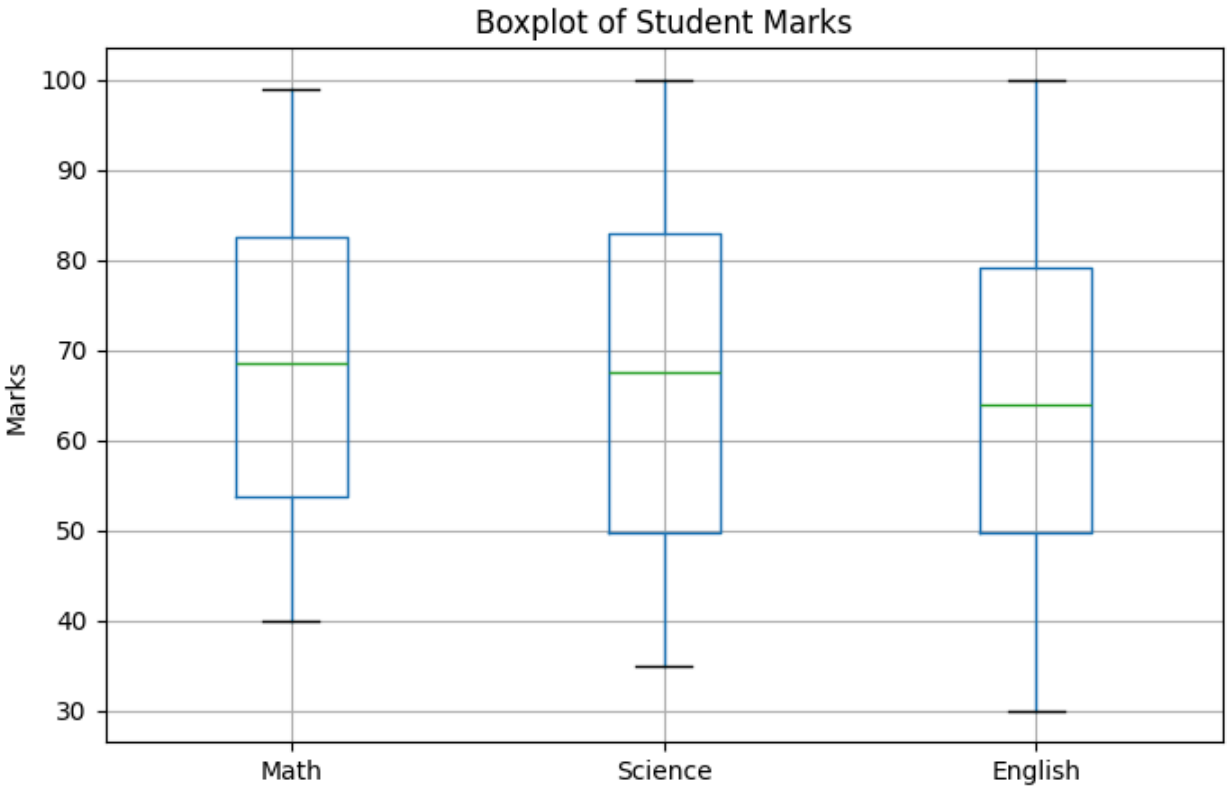
Part C - Case Study

In [133…

```
"""Case Study 1 – Student Performance Analytics
You are given data of 100 students containing their marks in Math, Science, ar
(a) Using Pandas, calculate the average marks for each student and identify th
(b) Compute subject-wise average and standard deviation; comment which subject
highest variation.
(c) Draw a boxplot for each subject and comment on outliers."""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Simulating the data for 100 students
np.random.seed(0)  # for reproducibility
data = {
    'Student': [f'Student_{i+1}' for i in range(100)],
    'Math': np.random.randint(40, 101, 100),
    'Science': np.random.randint(35, 101, 100),
    'English': np.random.randint(30, 101, 100)
}
df = pd.DataFrame(data)

# (a) Calculate average marks for each student and identify top 5 performers
df['Average'] = df[['Math', 'Science', 'English']].mean(axis=1)
top5 = df.nlargest(5, 'Average')
print("Top 5 Students based on Average Marks:\n", top5[['Student', 'Average']]

# (b) Compute subject-wise average and standard deviation
subject_stats = df[['Math', 'Science', 'English']].agg(['mean', 'std'])
print("\nSubject-wise Average and Standard Deviation:\n", subject_stats)
highest_variation_subject = subject_stats.loc['std'].idxmax()
print(f"\nSubject with highest variation: {highest_variation_subject}")
```

```
# (c) Draw a boxplot for each subject
plt.figure(figsize=(8,5))
df[['Math', 'Science', 'English']].boxplot()
plt.title("Boxplot of Student Marks")
plt.ylabel("Marks")
plt.show()

print("\nComment on outliers: Points outside the whiskers in the boxplot are c
```

```
Top 5 Students based on Average Marks:
        Student    Average
97  Student_98  92.333333
84  Student_85  91.333333
53  Student_54  88.666667
77  Student_78  87.000000
67  Student_68  86.666667

Subject-wise Average and Standard Deviation:
            Math    Science    English
mean   68.620000  66.860000  64.410000
std    18.117897  19.115819  19.664918

Subject with highest variation: English
```



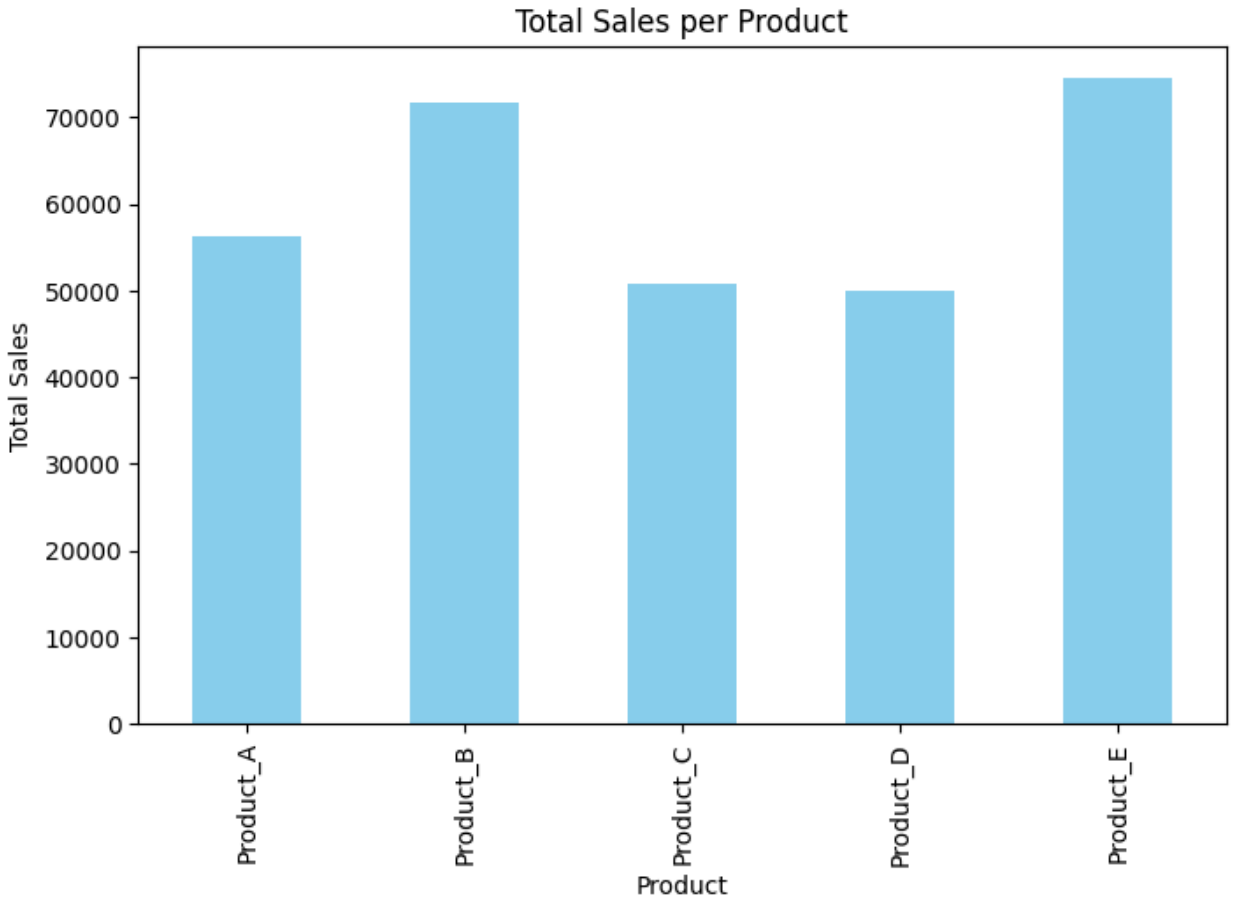

```
Comment on outliers: Points outside the whiskers in the boxplot are outliers, i
ndicating exceptionally low or high marks in that subject.
```

In [134…

```
"""Case Study 2 – Sales Data Exploration
A company records sales transactions in a dataset with columns: Product, Quant
Region.
(a) Add a new column Total = Quantity × Price and compute overall revenue.
```

```
(b) Group data by region and identify which region contributes the most sales.
(c) Plot a histogram of total sales per product and discuss which products are

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#Simulating sales data
np.random.seed(0)  # for reproducibility
products = ['Product_A', 'Product_B', 'Product_C', 'Product_D', 'Product_E']
regions = ['North', 'South', 'East', 'West']
data = {
    'Product': np.random.choice(products, 100),
    'Quantity': np.random.randint(1, 21, 100),
    'Price': np.random.randint(50, 501, 100),
    'Region': np.random.choice(regions, 100)
}
df = pd.DataFrame(data)

#(a)Add Total column and compute overall revenue
df['Total'] = df['Quantity'] * df['Price']
overall_revenue = df['Total'].sum()
print("Overall Revenue:", overall_revenue)

#(b)Group by region and identify top contributing region
region_sales = df.groupby('Region')['Total'].sum()
top_region = region_sales.idxmax()
print("\nSales by Region:\n", region_sales)
print(f"Region with highest sales: {top_region}")

#(c)Plot histogram of total sales per product
product_sales = df.groupby('Product')['Total'].sum()
plt.figure(figsize=(8,5))
product_sales.plot(kind='bar', color='skyblue')
plt.title("Total Sales per Product")
plt.ylabel("Total Sales")
plt.show()

print("\nComment on underperforming products: Products with significantly lowe
```

```
Overall Revenue: 303042

Sales by Region:
 Region
East     90107
North    90538
South    48222
West     74175
Name: Total, dtype: int64
Region with highest sales: North
```



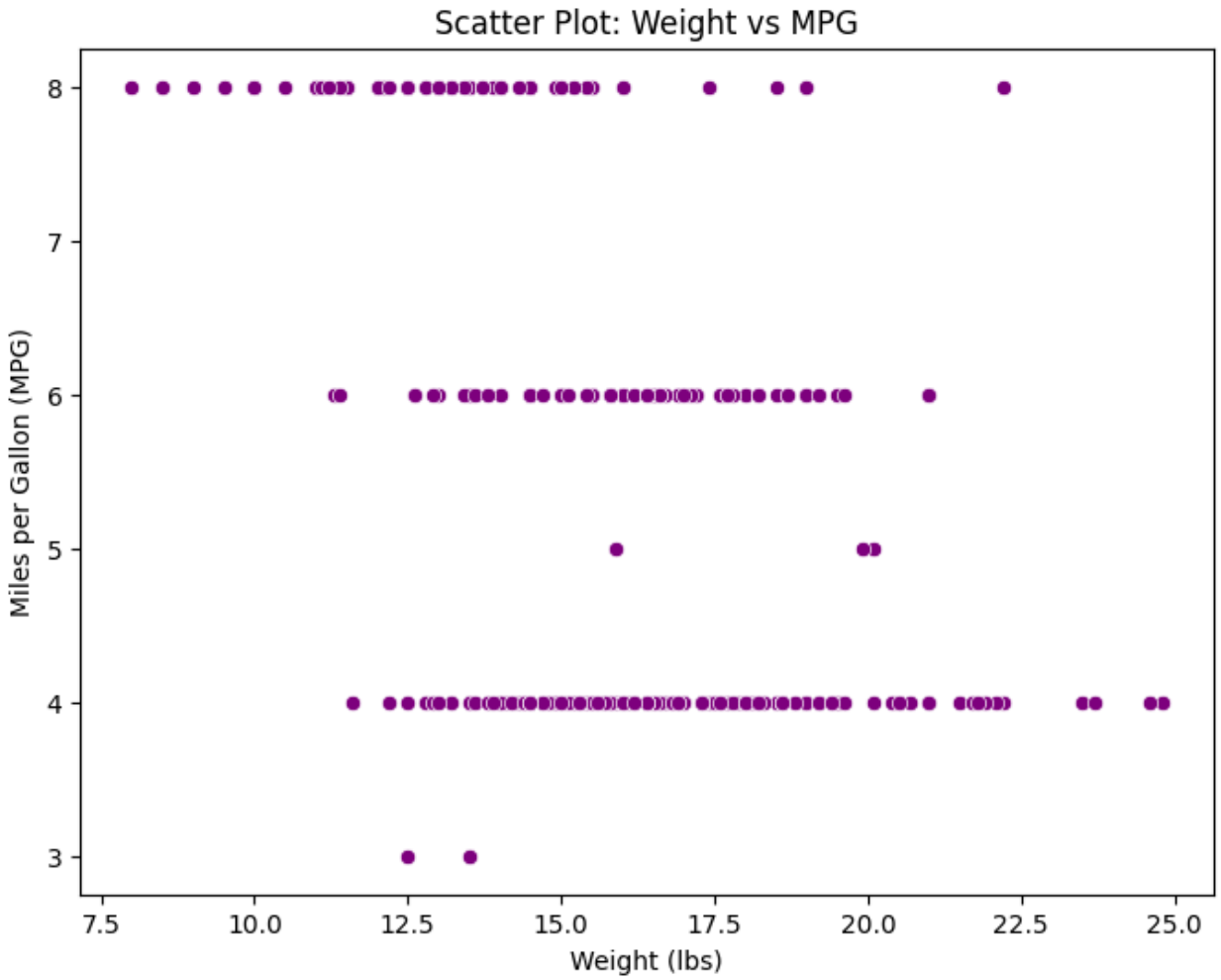
Comment on underperforming products: Products with significantly lower total sales compared to others may be underperforming.

In [135…

```
"""Case Study 3 – Predicting Fuel Efficiency (Auto MPG Dataset)
You are analyzing the Auto MPG dataset (UCI Repository) with features: mpg, cy
displacement, horsepower, weight, acceleration, model year, origin.
(a) Compute the correlation of mpg with each numeric feature and discuss which
influence fuel efficiency most.
(b) Plot a scatter plot of weight vs mpg; describe the trend you observe."""

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

#(a) Computing the correlation of mpg with each numeric feature and discuss wh
# Loading the Auto MPG dataset

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/autc
column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
                'acceleration', 'model_year', 'origin']

# Use raw string for regex separator and handle missing values
df = pd.read_csv(url, sep=r'\s+', names=column_names)
# Convert 'horsepower' to numeric, coercing errors to NaN
df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
```

```python

# Drop rows with NaN values in 'horsepower' after coercion
df.dropna(subset=['horsepower'], inplace=True)

# Now correlation will work
corr_matrix = df.corr(numeric_only=True) # Calculate correlation only on numer
print("Correlation with MPG:\n", corr_matrix['mpg'].sort_values(ascending=Fals

#(b) Scatter Plot of Weight vs MPG
# Scatter plot of weight vs mpg
plt.figure(figsize=(8, 6))
sns.scatterplot(x='weight', y='mpg', data=df, color='purple')
plt.title('Scatter Plot: Weight vs MPG')
plt.xlabel('Weight (lbs)')
plt.ylabel('Miles per Gallon (MPG)')
plt.show()
```

```
Correlation with MPG:
 mpg             1.000000
cylinders        0.950721
horsepower       0.896017
acceleration    -0.348746
weight          -0.505419
model_year      -0.562543
Name: mpg, dtype: float64
```


Scatter Plot: Weight vs MPG
Miles per Gallon (MPG)
Weight (lbs)
8
7
6
5
4
3
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0