Assignment – 1 (Chapter1-2)

Part A – Theory Questions

Q1. (a) From the dataset of a hospital (age, blood pressure, disease type, treatment success: Yes/No), mark which features are numerical and which are categorical.

Ans: Age and blood pressure are numerical features; disease type and treatment success are categorical.

(b) Why might using a categorical column (like disease type) as a number cause wrong conclusions?

Ans: Assigning numbers to categories like disease type may imply false order or magnitude (e.g., disease "A" = 1, "B" = 2 suggests B is greater), leading to misleading patterns.

(c) Which ML algorithm would naturally handle categorical data well? Which one fits numerical better?

Ans: Decision Trees and Random Forests handle categorical data well by splitting based on feature values. Linear Regression and Support Vector Machines are better suited for numerical data due to their reliance on mathematical relationships.

Q2. (a) A company records the salaries of employees. If one CEO earns 1 crore while most others earn below 50,000, would you use mean or median to describe central tendency? Why?

Ans: In this case, median is the better choice to describe central tendency. The CEO's extremely high salary (₹1 crore) is an outlier that would distort the mean, making it much higher than what most employees earn. The median, being the middle value, gives a more realistic picture of typical salaries.

(b) Define variance in your own words with an example.

Ans: Variance measures how much the data points differ from the average. It shows the spread of values. For example, if most employees earn around ₹40,000 but one earns ₹1 crore, the variance will be high, indicating large differences in salaries.

(c) How does standard deviation give more intuition about data spread compared to variance?

Ans: Standard deviation is the square root of variance and is expressed in the same unit as the data (e.g., rupees). This makes it easier to understand how far values typically deviate from the mean, giving a clearer sense of data spread than variance, which is in squared units.

Q3. (a) A student memorizes last year's question paper and scores well in mock tests but fails in the real exam. How is this similar to ML overfitting?

Ans: This is similar to **overfitting** in machine learning. The student memorizes specific questions rather than understanding the concepts. So, when faced with new questions in the real exam, they fail. Likewise, an overfitted ML model performs well on training data but poorly on unseen data because it has memorized patterns instead of learning general rules.

(b) List two strategies to improve generalization in models.

Ans: Two strategies to improve generalization are:

1. **Cross-validation** – testing the model on different subsets of data to ensure it performs consistently.
2. **Regularization** – adding penalties to overly complex models to prevent them from fitting noise in the data.

(c) Explain with an analogy why underfitting is as harmful as overfitting.

Ans: Imagine trying to fit clothes:

- **Overfitting** is like tailoring a suit too tightly to one person—it won't fit anyone else.
- **Underfitting** is like wearing a one-size-fits-all sack—it's too loose and doesn't suit anyone properly.
  Both fail to serve the purpose, just like poor ML models that either memorize or ignore patterns.

Q4. (a) A survey has missing "income" data for 10% of participants. Suggest two ways to address this issue.

Ans: Two ways to handle missing income data:

1. **Imputation** – Fill missing values using mean, median, or predictive models.
2. **Deletion** – Remove rows with missing data if the sample size remains sufficient and missingness is random.

(b) In a dataset of house prices, one record shows a price 100× higher than others. What is this called? How can it affect predictions?

Ans: A price 100× higher than others is an **outlier**. It can distort model training by skewing averages, increasing error, and leading to poor predictions, especially in regression models.

(c) Which is worse for a model — systematic missing values or random missing values? Explain briefly.

Ans: Systematic missing values are worse because they follow a pattern (e.g., missing income only for high earners), introducing bias. **Random missing values** are less harmful as they don't distort the data distribution significantly.

Q5. (a) A histogram of exam scores shows a long left tail. What does it suggest about student performance?

Ans: A histogram with a long left tail indicates **negative skewness**. Most students scored high, but a few scored very low. This suggests overall good performance, with a small group struggling significantly.

(b) A boxplot shows several dots above the whisker — what does that mean?

Ans: Dots above the upper whisker in a boxplot represent **outliers** — unusually high scores compared to the rest of the data. These could be exceptional performers or data errors.

(c) Why might scatter plots help you guess relationships before fitting a model?

Ans: Scatter plots visually show how two variables relate — whether they increase together, move oppositely, or show no pattern. This helps you **spot trends, clusters, or correlations** before applying any machine learning model.

Q6. (a) In predicting house prices, which would be the features and which is the target?

Ans: In predicting house prices, the **target** is the house price itself. The **features** are the input variables used to make the prediction, such as number of bedrooms, location, square footage, age of the house, and amenities.

(b) If one feature is strongly correlated with the target, how does it help the model?

Ans: If a feature is strongly correlated with the target, it helps the model by providing a reliable signal. For example, if square footage is highly correlated with price, the model can use it to make more accurate predictions, reducing error and improving performance.

(c) Can we ever have multiple targets? Give an example.

Ans: Yes, we can have **multiple targets** in multi-output regression or classification. For example, predicting both **house price** and **rental value** from the same set of features, or predicting **emotion** and **sentiment** from a text input.

Q7. (a) Classify these scenarios: grouping songs by similarity, predicting next month's rainfall, teaching a robot to balance.

Ans:

- **Grouping songs by similarity** → *Unsupervised learning* (no labels, clustering based on features).
- **Predicting next month's rainfall** → *Supervised learning* (uses past labeled data to predict future values).
- **Teaching a robot to balance** → *Reinforcement learning* (learns through trial and error with rewards).

(b) Why is supervised learning more common in industry?

Ans: Supervised learning is more common in industry because labeled data is often available, and the performance can be directly measured. It's reliable for tasks like fraud detection, demand forecasting, and customer churn prediction where clear input-output pairs exist.

(c) Give one benefit of unsupervised learning despite being harder to evaluate.

Ans: A key benefit of **unsupervised learning** is its ability to discover hidden patterns or groupings in data without needing labels. For example, it can segment customers into meaningful clusters, helping businesses tailor marketing strategies—even when no prior categories are defined.

Q8. (a) A recruitment dataset contains more male than female applicants. What type of data bias is this?

Ans: This is an example of **sampling bias** or **representation bias**, where one group (male applicants) is overrepresented in the dataset compared to others (female applicants).

(b) How can such bias affect ML model predictions?

Ans: Such bias can lead the ML model to learn patterns that favor the dominant group. For example, it may predict higher chances of selection for male candidates simply because it has seen more male data, resulting in unfair or discriminatory outcomes.

(c) Suggest one way to reduce the impact of data bias.
Ans: One way to reduce the impact is **rebalancing the dataset** — either by collecting more data from underrepresented groups or using techniques like **resampling** (oversampling minority class or undersampling majority class) to ensure fair representation during training.

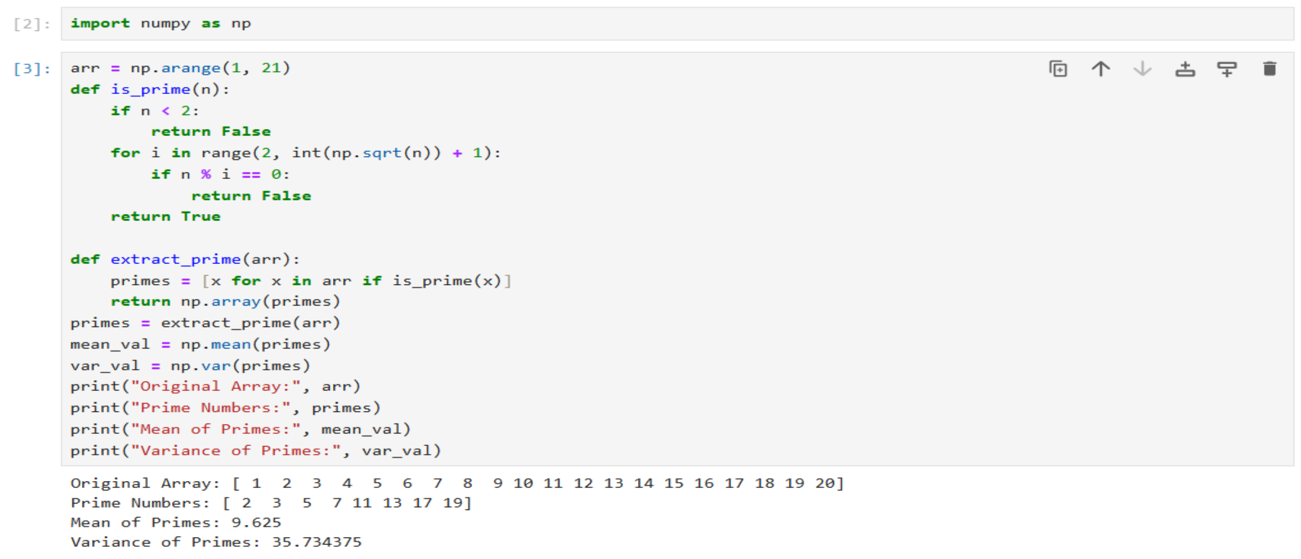
Q9. (a) If you have only 50 data points, would you use a very complex model or a simple one? Why?
Ans: With only 50 data points, it's better to use a **simple model**. Complex models need large datasets to learn meaningful patterns. On small data, they tend to memorize noise, leading to poor performance on new data.

(b) What danger arises if the model has too many parameters compared to the dataset size?
Ans: If the model has too many parameters compared to the dataset size, it risks **overfitting** — learning the training data too well, including random fluctuations, which reduces its ability to generalize to unseen data.

(c) Why do simpler models often generalize better on small datasets?
Ans: Simpler models often generalize better on small datasets because they focus on capturing the **core patterns** rather than fitting every detail. This helps avoid overfitting and improves performance on test or real-world data.

Part B – Lab Questions

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.

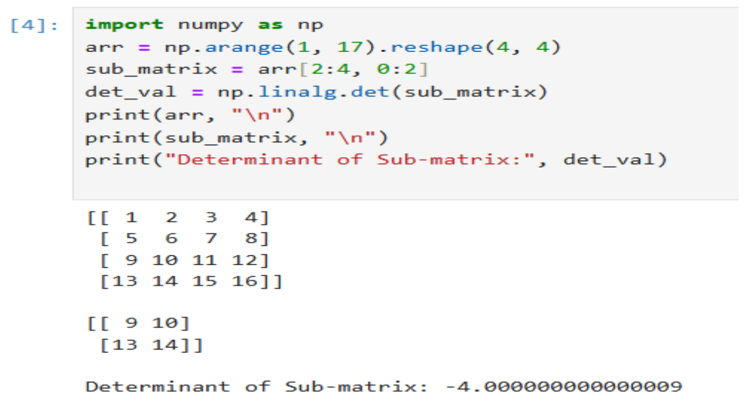


```
[2]: import numpy as np
```

```
[3]: arr = np.arange(1, 21)
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

def extract_prime(arr):
    primes = [x for x in arr if is_prime(x)]
    return np.array(primes)
primes = extract_prime(arr)
mean_val = np.mean(primes)
var_val = np.var(primes)
print("Original Array:", arr)
print("Prime Numbers:", primes)
print("Mean of Primes:", mean_val)
print("Variance of Primes:", var_val)
```

```
Original Array: [ 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
```

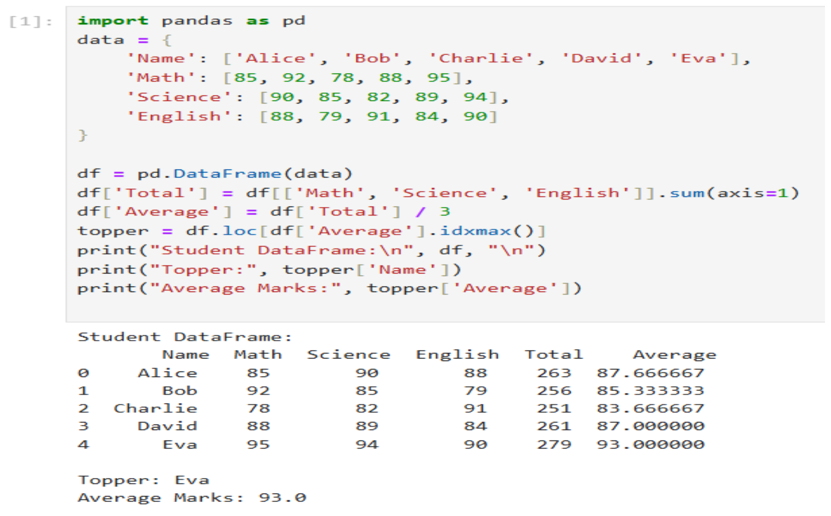
Q2. (a) Create a 4×4 NumPy array with numbers 1 to 16. (b) Extract the 2×2 bottom-left sub-matrix. (c) Compute the determinant of the sub-matrix.



```
[4]: import numpy as np
arr = np.arange(1, 17).reshape(4, 4)
sub_matrix = arr[2:4, 0:2]
det_val = np.linalg.det(sub_matrix)
print(arr, "\n")
print(sub_matrix, "\n")
print("Determinant of Sub-matrix:", det_val)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]

[[ 9 10]
 [13 14]]

Determinant of Sub-matrix: -4.000000000000009
```

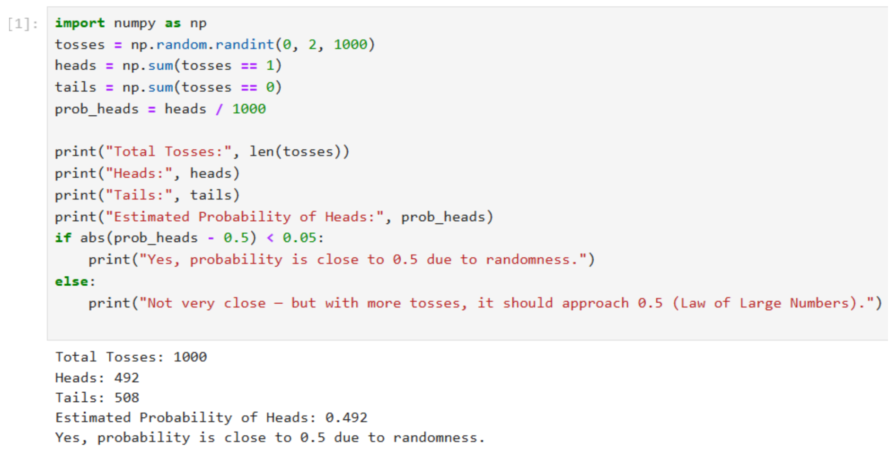
Q3. (a) Create a DataFrame with 5 students and marks in 3 subjects. (b) Add a column for total and average marks. (c) Identify the topper and print their name with average.



```
[1]: import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Math': [85, 92, 78, 88, 95],
    'Science': [90, 85, 82, 89, 94],
    'English': [88, 79, 91, 84, 90]
}

df = pd.DataFrame(data)
df['Total'] = df[['Math', 'Science', 'English']].sum(axis=1)
df['Average'] = df['Total'] / 3
topper = df.loc[df['Average'].idxmax()]
print("Student DataFrame:\n", df, "\n")
print("Topper:", topper['Name'])
print("Average Marks:", topper['Average'])
```

```
Student DataFrame:
       Name  Math  Science  English  Total    Average
0    Alice    85       90       88    263  87.666667
1      Bob    92       85       79    256  85.333333
2  Charlie    78       82       91    251  83.666667
3    David    88       89       84    261  87.000000
4      Eva    95       94       90    279  93.000000

Topper: Eva
Average Marks: 93.0
```

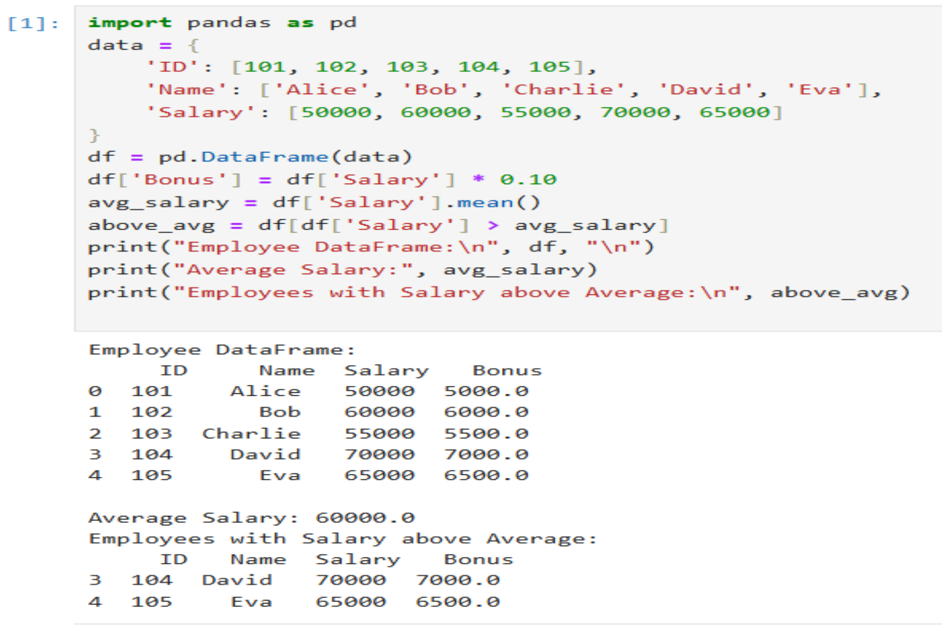
Q4. (a) Simulate 1000 coin tosses using NumPy (1=Head, 0=Tail). (b) Count frequency of heads and tails. (c) Estimate probability of heads. Is it close to 0.5? Why/Why not?



```
[1]: import numpy as np
     tosses = np.random.randint(0, 2, 1000)
     heads = np.sum(tosses == 1)
     tails = np.sum(tosses == 0)
     prob_heads = heads / 1000

     print("Total Tosses:", len(tosses))
     print("Heads:", heads)
     print("Tails:", tails)
     print("Estimated Probability of Heads:", prob_heads)
     if abs(prob_heads - 0.5) < 0.05:
         print("Yes, probability is close to 0.5 due to randomness.")
     else:
         print("Not very close – but with more tosses, it should approach 0.5 (Law of Large Numbers).")
```

```
Total Tosses: 1000
Heads: 492
Tails: 508
Estimated Probability of Heads: 0.492
Yes, probability is close to 0.5 due to randomness.
```

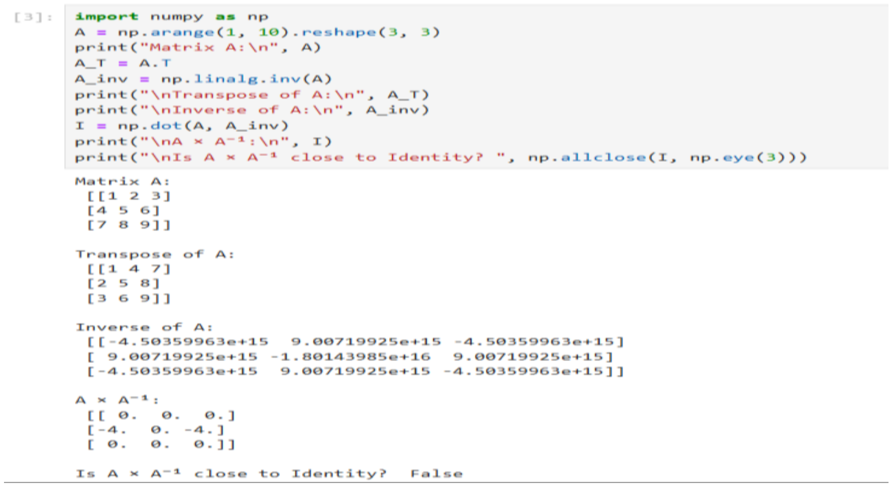
Q5. (a) Create a DataFrame of employees with columns: ID, Name, Salary. (b) Add a Bonus column = 10% of Salary.(c) Display employees with salary above average.



```
[1]: import pandas as pd
     data = {
         'ID': [101, 102, 103, 104, 105],
         'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
         'Salary': [50000, 60000, 55000, 70000, 65000]
     }
     df  = pd.DataFrame(data)
     df['Bonus'] = df['Salary'] * 0.10
     avg_salary = df['Salary'].mean()
     above_avg = df[df['Salary'] > avg_salary]
     print("Employee DataFrame:\n", df, "\n")
     print("Average Salary:", avg_salary)
     print("Employees with Salary above Average:\n", above_avg)
```

```
Employee DataFrame:
      ID     Name  Salary   Bonus
0   101    Alice   50000  5000.0
1   102      Bob   60000  6000.0
2   103  Charlie   55000  5500.0
3   104    David   70000  7000.0
4   105      Eva   65000  6500.0

Average Salary: 60000.0
Employees with Salary above Average:
      ID   Name  Salary   Bonus
3   104  David   70000  7000.0
4   105    Eva   65000  6500.0
```

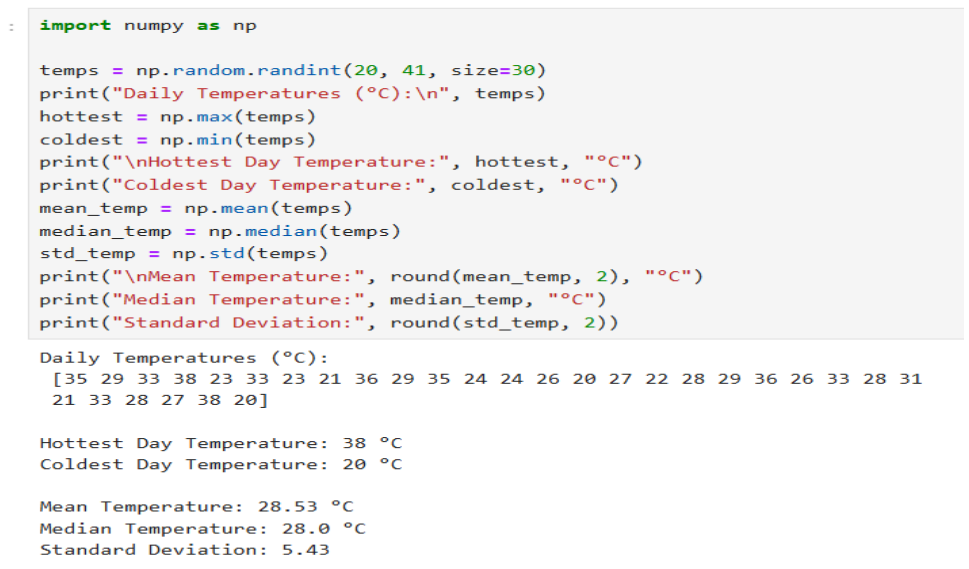
Q6. (a) Create a 3×3 NumPy array with values 1 to 9. (b) Find its transpose and inverse. (c) Verify that A × A^{-1} ≈ I.



```
[3]: import numpy as np
     A = np.arange(1, 10).reshape(3, 3)
     print("Matrix A:\n", A)
     A_T = A.T
     A_inv = np.linalg.inv(A)
     print("\nTranspose of A:\n", A_T)
     print("\nInverse of A:\n", A_inv)
     I = np.dot(A, A_inv)
     print("\nA × A⁻¹:\n", I)
     print("\nIs A × A⁻¹ close to Identity? ", np.allclose(I, np.eye(3)))
```

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

Transpose of A:
 [[1 4 7]
 [2 5 8]
 [3 6 9]]

Inverse of A:
 [[-4.50359963e+15  9.00719925e+15 -4.50359963e+15]
 [ 9.00719925e+15 -1.80143985e+16  9.00719925e+15]
 [-4.50359963e+15  9.00719925e+15 -4.50359963e+15]]

A × A⁻¹:
 [[ 0.   0.   0.]
 [-4.   0.  -4.]
 [ 0.   0.   0.]]

Is A × A⁻¹ close to Identity?  False
```

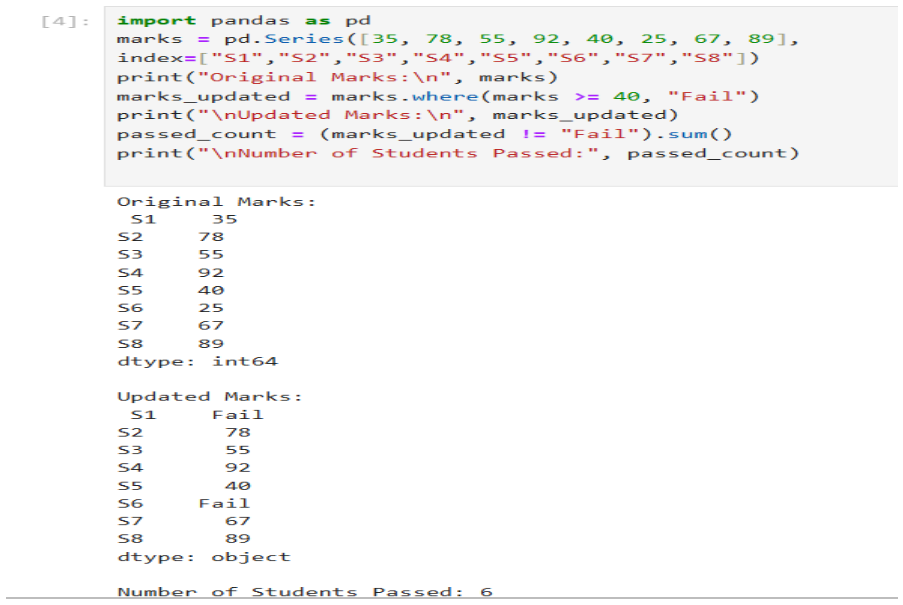
Q7. (a) Generate random daily temperatures (30 values, range 20–40°C). (b) Find the hottest and coldest day. (c) Compute mean, median, and standard deviation of temperatures.



```python
import numpy as np

temps = np.random.randint(20, 41, size=30)
print("Daily Temperatures (°C):\n", temps)
hottest = np.max(temps)
coldest = np.min(temps)
print("\nHottest Day Temperature:", hottest, "°C")
print("Coldest Day Temperature:", coldest, "°C")
mean_temp = np.mean(temps)
median_temp = np.median(temps)
std_temp = np.std(temps)
print("\nMean Temperature:", round(mean_temp, 2), "°C")
print("Median Temperature:", median_temp, "°C")
print("Standard Deviation:", round(std_temp, 2))
```

```
Daily Temperatures (°C):
 [35 29 33 38 23 33 23 21 36 29 35 24 24 26 20 27 22 28 29 36 26 33 28 31
 21 33 28 27 38 20]

Hottest Day Temperature: 38 °C
Coldest Day Temperature: 20 °C

Mean Temperature: 28.53 °C
Median Temperature: 28.0 °C
Standard Deviation: 5.43
```

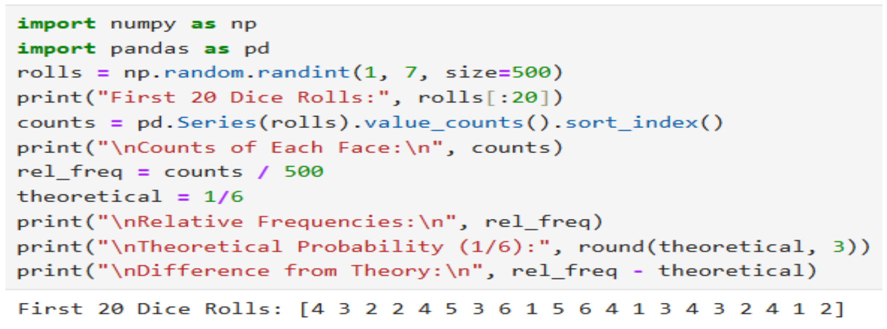
Q8. (a) Create a Pandas Series of marks for 8 students. (b) Replace all marks below 40 with “Fail”. (c) Count how many students passed.



```python
import pandas as pd
marks = pd.Series([35, 78, 55, 92, 40, 25, 67, 89],
index=["S1","S2","S3","S4","S5","S6","S7","S8"])
print("Original Marks:\n", marks)
marks_updated = marks.where(marks >= 40, "Fail")
print("\nUpdated Marks:\n", marks_updated)
passed_count = (marks_updated != "Fail").sum()
print("\nNumber of Students Passed:", passed_count)
```

```
Original Marks:
 S1    35
S2    78
S3    55
S4    92
S5    40
S6    25
S7    67
S8    89
dtype: int64

Updated Marks:
 S1    Fail
S2      78
S3      55
S4      92
S5      40
S6    Fail
S7      67
S8      89
dtype: object

Number of Students Passed: 6
```

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.



```python
import numpy as np
import pandas as pd
rolls = np.random.randint(1, 7, size=500)
print("First 20 Dice Rolls:", rolls[:20])
counts = pd.Series(rolls).value_counts().sort_index()
print("\nCounts of Each Face:\n", counts)
rel_freq = counts / 500
theoretical = 1/6
print("\nRelative Frequencies:\n", rel_freq)
print("\nTheoretical Probability (1/6):", round(theoretical, 3))
print("\nDifference from Theory:\n", rel_freq - theoretical)
```

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

```
Counts of Each Face:
 1      75
2      81
3     103
4      75
5      96
6      70
Name: count, dtype: int64

Relative Frequencies:
 1     0.150
2     0.162
3     0.206
4     0.150
5     0.192
6     0.140
Name: count, dtype: float64

Theoretical Probability (1/6): 0.167
```

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.

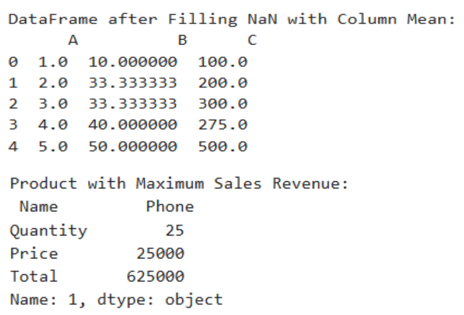
```
[6]: import pandas as pd
     data = {
         "Name": ["Laptop", "Phone", "Tablet", "Headphones", "Camera", "Smartwatch"],
         "Quantity": [10, 25, 15, 30, 12, 20],
         "Price": [60000, 25000, 15000, 2000, 40000, 12000]
     }
     df = pd.DataFrame(data)
     print("Product Data:\n", df)
     df["Total"] = df["Quantity"] * df["Price"]
     print("\nData with Total Sales:\n", df)
     max_revenue_product = df.loc[df["Total"].idxmax()]
     print("\nProduct with Maximum Sales Revenue:\n", max_revenue_product)
```

```
Product Data:
          Name  Quantity  Price
0      Laptop        10  60000
1       Phone        25  25000
2      Tablet        15  15000
3  Headphones        30   2000
4      Camera        12  40000
5  Smartwatch        20  12000

Data with Total Sales:
          Name  Quantity  Price   Total
0      Laptop        10  60000  600000
1       Phone        25  25000  625000
2      Tablet        15  15000  225000
3  Headphones        30   2000   60000
4      Camera        12  40000  480000
5  Smartwatch        20  12000  240000

Product with Maximum Sales Revenue:
 Name         Phone
Quantity        25
Price        25000
Total       625000
Name: 1, dtype: object
```

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.

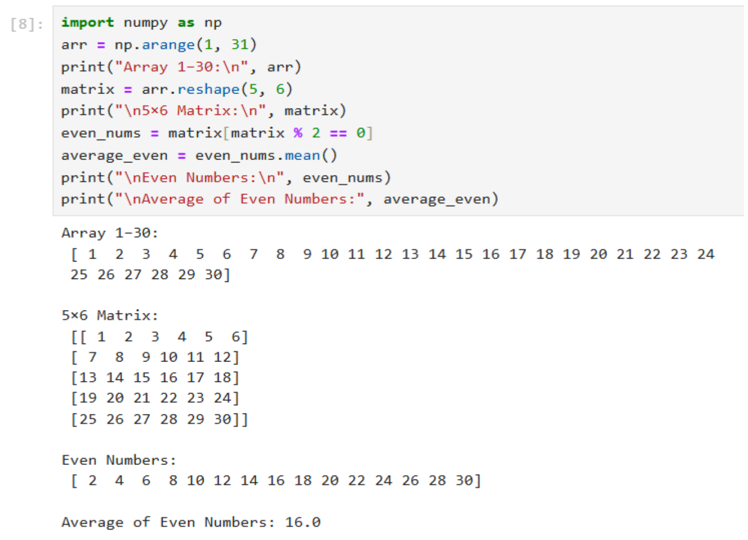
```
[7]: import pandas as pd
     import numpy as np
     data = {
         "A": [1, 2, np.nan, 4, 5],
         "B": [10, np.nan, np.nan, 40, 50],
         "C": [100, 200, 300, np.nan, 500]
     }
     df = pd.DataFrame(data)
     print("Original DataFrame:\n", df)
     df_filled = df.fillna(df.mean(numeric_only=True))
     print("\nDataFrame after Filling NaN with Column Mean:\n", df_filled)
     df_dropped = df.dropna(thresh=df.shape[1]-1)
     print("\nDataFrame after Dropping Rows with >1 Missing Values:\n", df_dropped)
```

```
Original DataFrame:
      A     B      C
0  1.0  10.0  100.0
1  2.0   NaN  200.0
2  NaN   NaN  300.0
3  4.0  40.0    NaN
4  5.0  50.0  500.0
```



```
DataFrame after Filling NaN with Column Mean:
      A          B      C
0  1.0  10.000000  100.0
1  2.0  33.333333  200.0
2  3.0  33.333333  300.0
3  4.0  40.000000  275.0
4  5.0  50.000000  500.0

Product with Maximum Sales Revenue:
 Name         Phone
Quantity        25
Price        25000
Total       625000
Name: 1, dtype: object
```

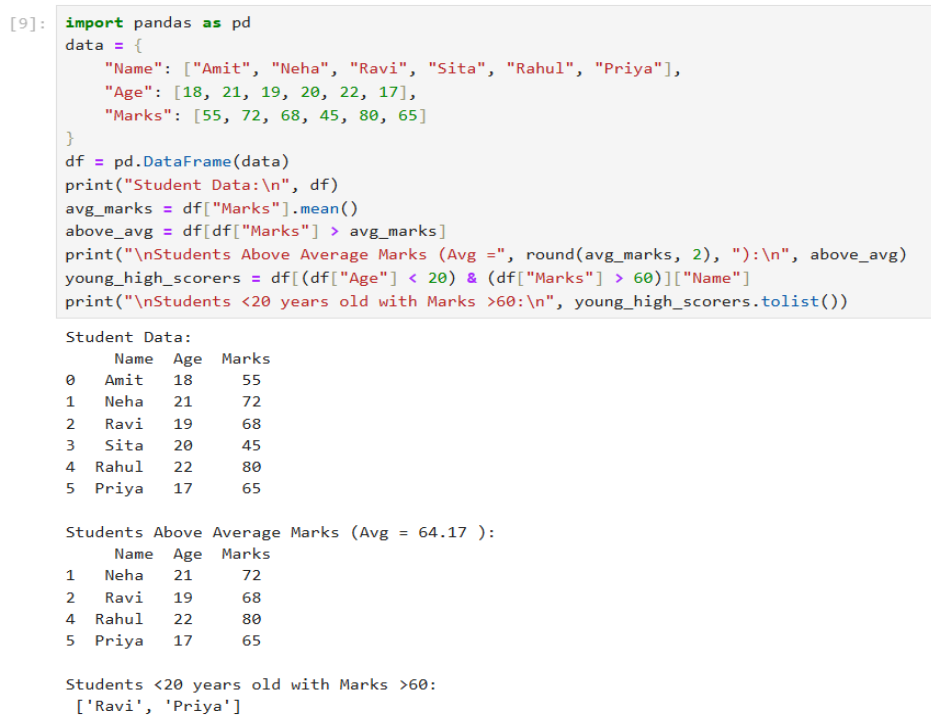
Q12. (a) Create a NumPy array with integers from 1 to 30. (b) Reshape it into a 5×6 matrix. (c) Extract all even numbers from the matrix and compute their average.



```
[8]: import numpy as np
     arr = np.arange(1, 31)
     print("Array 1-30:\n", arr)
     matrix = arr.reshape(5, 6)
     print("\n5×6 Matrix:\n", matrix)
     even_nums = matrix[matrix % 2 == 0]
     average_even = even_nums.mean()
     print("\nEven Numbers:\n", even_nums)
     print("\nAverage of Even Numbers:", average_even)
```

```
Array 1-30:
 [ 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]

5×6 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
```

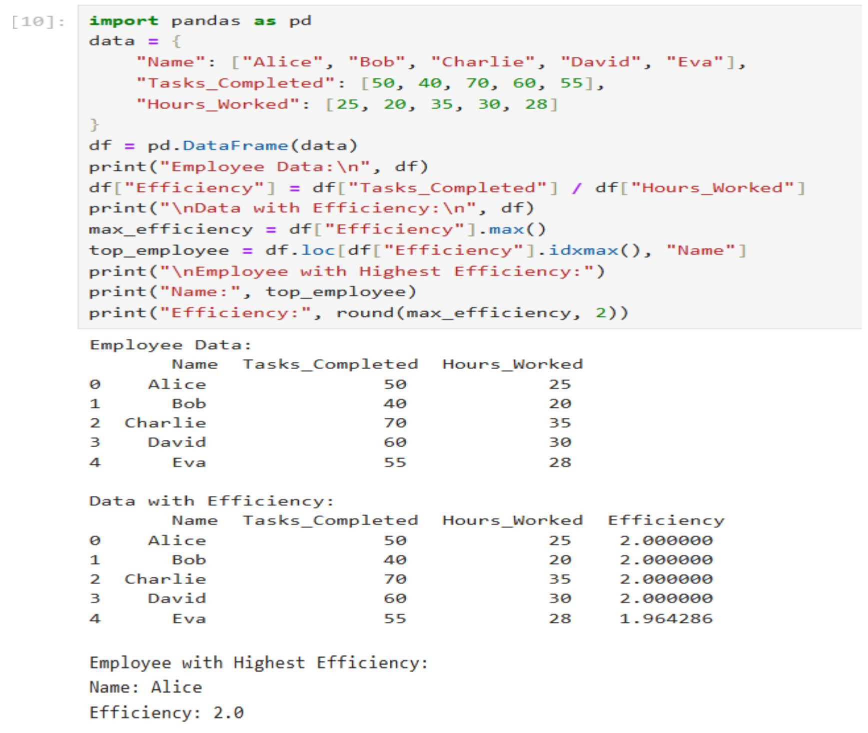
Q13. (a) Create a DataFrame of 6 students with columns: Name, Age, Marks. (b) Select only students who scored above the overall average marks. (c) Display names of students younger than 20 whose marks are above 60.



```
[9]: import pandas as pd
     data = {
         "Name": ["Amit", "Neha", "Ravi", "Sita", "Rahul", "Priya"],
         "Age": [18, 21, 19, 20, 22, 17],
         "Marks": [55, 72, 68, 45, 80, 65]
     }
     df = pd.DataFrame(data)
     print("Student Data:\n", df)
     avg_marks = df["Marks"].mean()
     above_avg = df[df["Marks"] > avg_marks]
     print("\nStudents Above Average Marks (Avg =", round(avg_marks, 2), "):\n", above_avg)
     young_high_scorers = df[(df["Age"] < 20) & (df["Marks"] > 60)]["Name"]
     print("\nStudents <20 years old with Marks >60:\n", young_high_scorers.tolist())
```

```
Student Data:
     Name  Age  Marks
0   Amit   18     55
1   Neha   21     72
2   Ravi   19     68
3   Sita   20     45
4  Rahul   22     80
5  Priya   17     65

Students Above Average Marks (Avg = 64.17 ):
     Name  Age  Marks
1   Neha   21     72
2   Ravi   19     68
4  Rahul   22     80
5  Priya   17     65

Students <20 years old with Marks >60:
 ['Ravi', 'Priya']
```

Q14. (a) Create a Pandas DataFrame with 5 employees having columns: Name, Tasks_Completed, 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 value.



```
[10]:
import pandas as pd
data = {
    "Name": ["Alice", "Bob", "Charlie", "David", "Eva"],
    "Tasks_Completed": [50, 40, 70, 60, 55],
    "Hours_Worked": [25, 20, 35, 30, 28]
}
df = pd.DataFrame(data)
print("Employee Data:\n", df)
df["Efficiency"] = df["Tasks_Completed"] / df["Hours_Worked"]
print("\nData with Efficiency:\n", df)
max_efficiency = df["Efficiency"].max()
top_employee = df.loc[df["Efficiency"].idxmax(), "Name"]
print("\nEmployee with Highest Efficiency:")
print("Name:", top_employee)
print("Efficiency:", round(max_efficiency, 2))
```

```
Employee Data:
       Name  Tasks_Completed  Hours_Worked
0    Alice               50            25
1      Bob               40            20
2  Charlie               70            35
3    David               60            30
4      Eva               55            28

Data with Efficiency:
       Name  Tasks_Completed  Hours_Worked  Efficiency
0    Alice               50            25    2.000000
1      Bob               40            20    2.000000
2  Charlie               70            35    2.000000
3    David               60            30    2.000000
4      Eva               55            28    1.964286

Employee with Highest Efficiency:
Name: Alice
Efficiency: 2.0
```

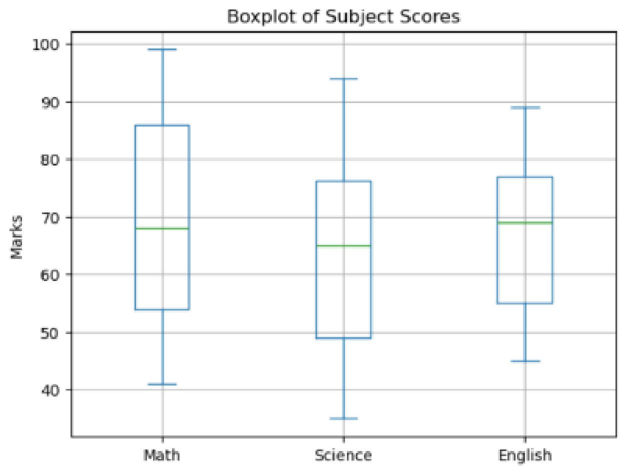
Part C – Case Study

Case Study 1 – Student Performance Analytics

You are given data of 100 students containing their marks in **Math, Science, and English**. (a) Using Pandas, calculate the average marks for each student and identify the top 5 performers. (b) Compute subject-wise average and standard deviation; comment which subject shows the highest variation. (c) Draw a boxplot for each subject and comment on outliers.

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

np.random.seed(42)
df = pd.DataFrame({
    'Student': [f'Student_{i+1}' for i in range(100)],
    'Math': np.random.randint(40, 100, 100),
    'Science': np.random.randint(35, 95, 100),
    'English': np.random.randint(45, 90, 100)
})

df['Average'] = df[['Math', 'Science', 'English']].mean(axis=1)
top_5 = df.sort_values(by='Average', ascending=False).head(5)
print("Top 5 Performers:\n", top_5[['Student', 'Average']])

subject_stats = df[['Math', 'Science', 'English']].agg(['mean', 'std'])
print("\nSubject-wise Mean and Standard Deviation:\n", subject_stats)
highest_var_subject = subject_stats.loc['std'].idxmax()
print(f"\nSubject with highest variation: {highest_var_subject}")

plt.figure(figsize=(8,6))
df[['Math', 'Science', 'English']].plot(kind='box')
plt.title('Boxplot of Subject Scores')
plt.ylabel('Marks')
plt.grid(True)
plt.show()

for subject in ['Math', 'Science', 'English']:
    q1 = df[subject].quantile(0.25)
    q3 = df[subject].quantile(0.75)
    iqr = q3 - q1
    outliers = df[(df[subject] < q1 - 1.5 * iqr) | (df[subject] > q3 + 1.5 * iqr)]
    print(f"\n{subject} has {len(outliers)} outlier(s).")
```

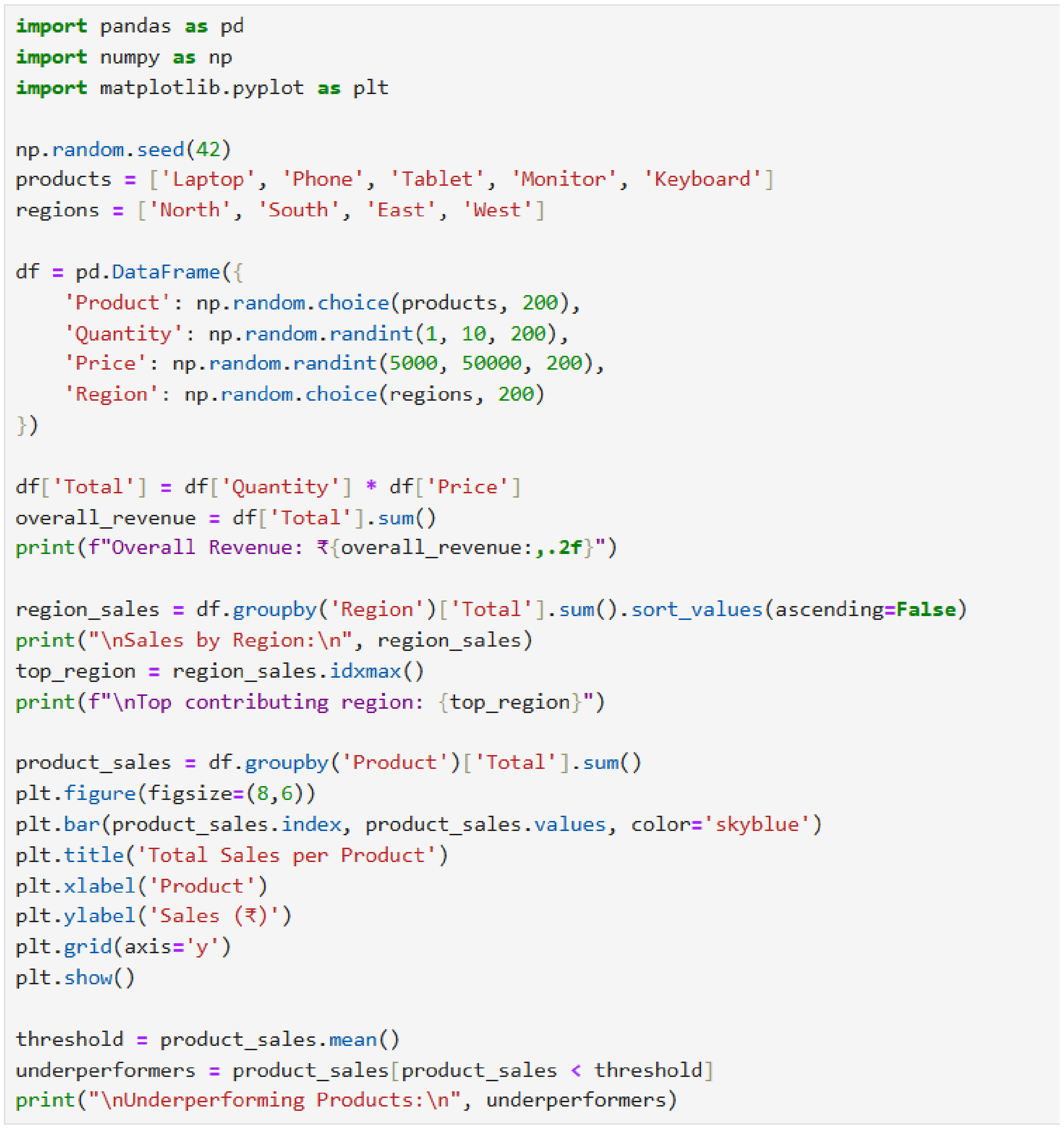
```
Top 5 Performers:
         Student    Average
14   Student_15  86.000000
16   Student_17  85.333333
65   Student_66  83.666667
37   Student_38  83.333333
85   Student_86  83.000000

Subject-wise Mean and Standard Deviation:
             Math    Science    English
mean  69.580000  64.070000  67.120000
std   18.031499  16.811645  13.556675

Subject with highest variation: Math
<Figure size 800x600 with 0 Axes>
```



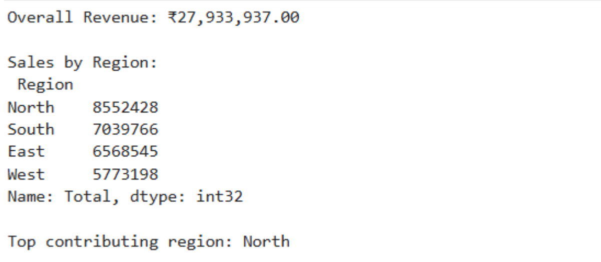
```
Math has 0 outlier(s).

Science has 0 outlier(s).

English has 0 outlier(s).
```

Case Study 2 – Sales Data Exploration

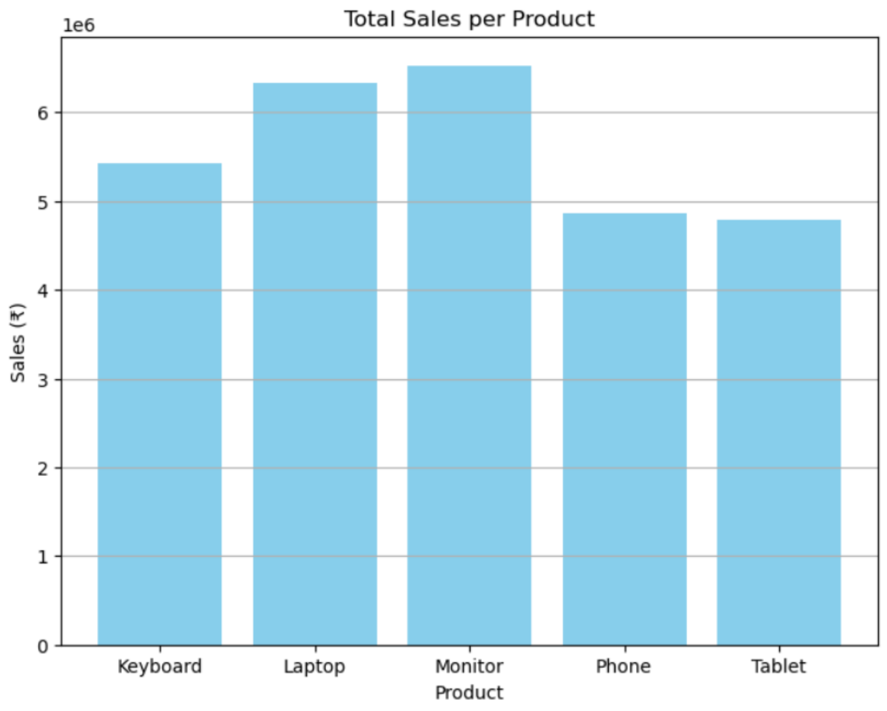
A company records sales transactions in a dataset with columns: **Product, Quantity, Price, Region**. (a) Add a new column Total = Quantity × Price and compute overall revenue. (b) Group data by region and identify which region contributes the most sales. (c) Plot a histogram of total sales per product and discuss which products are underperforming.



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

np.random.seed(42)
products = ['Laptop', 'Phone', 'Tablet', 'Monitor', 'Keyboard']
regions = ['North', 'South', 'East', 'West']

df = pd.DataFrame({
    'Product': np.random.choice(products, 200),
    'Quantity': np.random.randint(1, 10, 200),
    'Price': np.random.randint(5000, 50000, 200),
    'Region': np.random.choice(regions, 200)
})

df['Total'] = df['Quantity'] * df['Price']
overall_revenue = df['Total'].sum()
print(f"Overall Revenue: ₹{overall_revenue:,.2f}")

region_sales = df.groupby('Region')['Total'].sum().sort_values(ascending=False)
print("\nSales by Region:\n", region_sales)
top_region = region_sales.idxmax()
print(f"\nTop contributing region: {top_region}")

product_sales = df.groupby('Product')['Total'].sum()
plt.figure(figsize=(8,6))
plt.bar(product_sales.index, product_sales.values, color='skyblue')
plt.title('Total Sales per Product')
plt.xlabel('Product')
plt.ylabel('Sales (₹)')
plt.grid(axis='y')
plt.show()

threshold = product_sales.mean()
underperformers = product_sales[product_sales < threshold]
print("\nUnderperforming Products:\n", underperformers)
```



```
Overall Revenue: ₹27,933,937.00

Sales by Region:
 Region
North    8552428
South    7039766
East     6568545
West     5773198
Name: Total, dtype: int32

Top contributing region: North
```



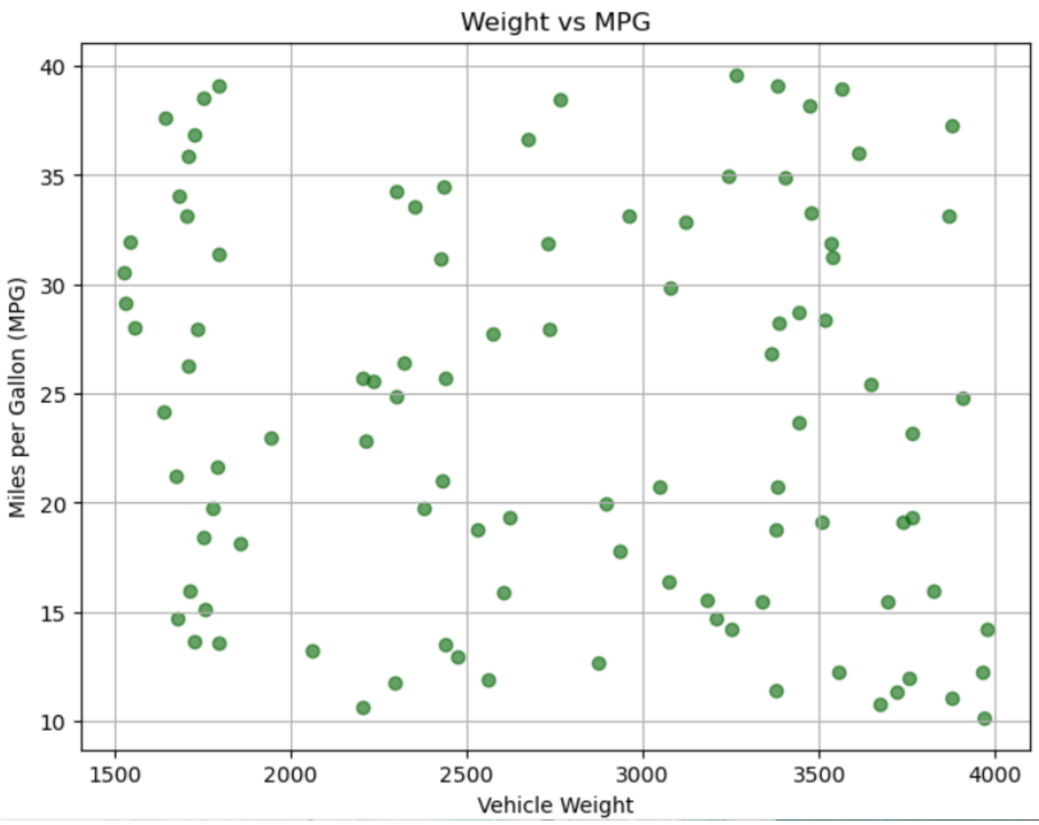
```
Underperforming Products:
 Product
Keyboard    5420283
Phone       4868416
Tablet      4792354
Name: Total, dtype: int32
```

Case Study 3 – Predicting Fuel Efficiency (Auto MPG Dataset)

You are analyzing the **Auto MPG dataset (UCI Repository)** with features: mpg, cylinders, displacement, horsepower, weight, acceleration, model year, origin. (a) Compute the correlation of mpg with each numeric feature and discuss which factors influence fuel efficiency most. (b) Plot a scatter plot of weight vs mpg; describe the trend you observe.

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

np.random.seed(42)
df = pd.DataFrame({
    'mpg': np.random.uniform(10, 40, 100),
    'cylinders': np.random.choice([4, 6, 8], 100),
    'displacement': np.random.uniform(100, 450, 100),
    'horsepower': np.random.uniform(50, 200, 100),
    'weight': np.random.uniform(1500, 4000, 100),
    'acceleration': np.random.uniform(8, 20, 100),
    'model_year': np.random.randint(70, 83, 100),
    'origin': np.random.choice([1, 2, 3], 100)
})

numeric_features = ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model_year']
correlations = df[numeric_features + ['mpg']].corr()['mpg'].drop('mpg')
print("Correlation of mpg with other features:\n", correlations)

most_negative = correlations.idxmin()
most_positive = correlations.idxmax()
print(f"\nMost negative correlation: {most_negative} → heavier/larger engines reduce fuel efficiency.")
print(f"Most positive correlation: {most_positive} → newer models tend to be more fuel-efficient.")

plt.figure(figsize=(8,6))
plt.scatter(df['weight'], df['mpg'], color='darkgreen', alpha=0.6)
plt.title('Weight vs MPG')
plt.xlabel('Vehicle Weight')
plt.ylabel('Miles per Gallon (MPG)')
plt.grid(True)
plt.show()

print("\nTrend Observation:")
print("As vehicle weight increases, MPG tends to decrease — heavier cars are generally less fuel-efficient.")
```

```
Correlation of mpg with other features:
 cylinders      -0.043319
displacement    0.067325
horsepower      0.125899
weight         -0.134264
acceleration   -0.138451
model_year     -0.006295
Name: mpg, dtype: float64

Most negative correlation: acceleration → heavier/larger engines reduce fuel efficiency.
Most positive correlation: horsepower → newer models tend to be more fuel-efficient.
```



```
Trend Observation:
As vehicle weight increases, MPG tends to decrease — heavier cars are generally less fuel-efficient.
```