Python-230901103 / Skill test assignment 1-230901103(1).ipynb

Sundar-Sabari Add files via upload 999cb1e · 7 minutes ago

1041 lines (1041 loc) · 444 KB

Preview Code Blame Raw

1A) Analyse student exam scores with python

In [1]:
```
import pandas as pd
import numpy as np

# Sample exam scores of 20 students
scores = [72, 45, 88, 60, 90, 33, 55, 67, 79, 48, 95, 58, 66, 82, 74, 39, 50, 6

# Create a Pandas DataFrame
df = pd.DataFrame({'Student_ID': range(1, 21), 'Score': scores})

# 1. Overall class performance
average_score = np.mean(df['Score'])
highest_score = np.max(df['Score'])
lowest_score = np.min(df['Score'])

print("Overall Class Performance:")
print(f"Average Score: {average_score:.2f}")
print(f"Highest Score: {highest_score}")
print(f"Lowest Score: {lowest_score}\n")

# 2. Pass/fail classification
df['Result'] = np.where(df['Score'] >= 50, 'Pass', 'Fail')

# 3. Grading
def assign_grade(score):
    if score >= 85:
        return 'A'
    elif score >= 70:
        return 'B'
    elif score >= 55:
        return 'C'
    elif score >= 50:
        return 'D'
    else:
        return 'F'

df['Grade'] = df['Score'].apply(assign_grade)

# Display the final DataFrame
print("Student Performance Analysis:\n")
print(df)
```

```
Overall Class Performance:
Average Score: 64.10
Highest Score: 95
Lowest Score: 33

Student Performance Analysis:

    Student_ID  Score Result Grade
0            1     72   Pass     B
1            2     45   Fail     F
2            3     88   Pass     A
3            4     60   Pass     C
```

```
4            5     90   Pass     A
5            6     33   Fail     F
6            7     55   Pass     C
7            8     67   Pass     C
8            9     79   Pass     B
9           10     48   Fail     F
10          11     95   Pass     A
11          12     58   Pass     C
12          13     66   Pass     C
13          14     82   Pass     B
14          15     74   Pass     B
15          16     39   Fail     F
16          17     50   Pass     D
17          18     61   Pass     C
18          19     77   Pass     B
19          20     43   Fail     F
```

1B)Solve a system of linear equations

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

# Step 1: Define matrix A and vector b
A = np.array([[2, 3],
              [1, 2]])

b = np.array([8, 5])

# Step 2: Solve the system Ax = b
x = np.linalg.solve(A, b)

# Step 3: Display the solution
print("Solution vector x:")
print(x)
```

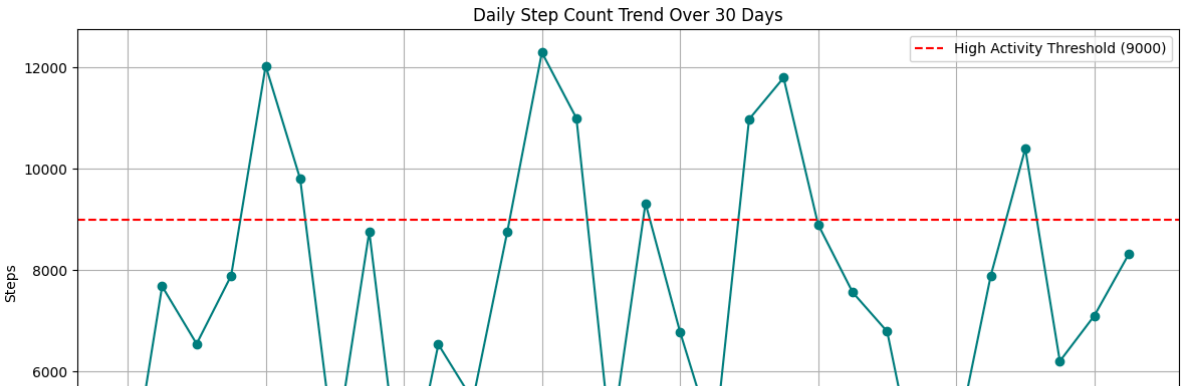
```
Solution vector x:
[1. 2.]
```

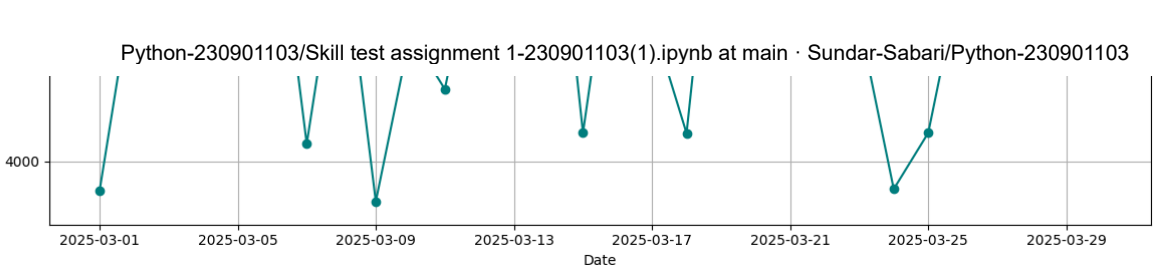
2A)Analyze daily step using numpy

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Sample daily step count data for 30 days
# You can replace this with your own data
daily_steps = np.array([
    3421, 7680, 6543, 7890, 12034, 9800, 4350,
    8760, 3210, 6532, 5430, 8765, 12300, 11000,
    4567, 9321, 6780, 4560, 10980, 11800, 8900,
    7560, 6785, 3456, 4567, 7890, 10400, 6200, 7100, 8320
])

# Create a Pandas DataFrame with dates
dates = pd.date_range(start="2025-03-01", periods=30)
df = pd.DataFrame({'Date': dates, 'Steps': daily_steps})

# 1. Key statistics
```

```
total_steps = np.sum(daily_steps)
average_steps = np.mean(daily_steps)
max_steps = np.max(daily_steps)
min_steps = np.min(daily_steps)

print("=== Step Count Statistics ===")
print(f"Total steps: {total_steps}")
print(f"Average steps per day: {average_steps:.2f}")
print(f"Maximum steps in a day: {max_steps}")
print(f"Minimum steps in a day: {min_steps}")

# 2. Identify high activity days (e.g., more than 9000 steps)
threshold = 9000
high_activity_days = df[df['Steps'] > threshold]
print("\n=== High Activity Days (Steps > 9000) ===")
print(high_activity_days)

# 3. Analyze trend (basic: plot steps over time)
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Steps'], marker='o', linestyle='-', color='teal')
plt.title('Daily Step Count Trend Over 30 Days')
plt.xlabel('Date')
plt.ylabel('Steps')
plt.axhline(y=threshold, color='red', linestyle='--', label='High Activity Thres
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
=== Step Count Statistics ===
Total steps: 226901
Average steps per day: 7563.37
Maximum steps in a day: 12300
Minimum steps in a day: 3210

=== High Activity Days (Steps > 9000) ===
         Date  Steps
4  2025-03-05  12034
5  2025-03-06   9800
12 2025-03-13  12300
13 2025-03-14  11000
15 2025-03-16   9321
18 2025-03-19  10980
19 2025-03-20  11800
26 2025-03-27  10400
```





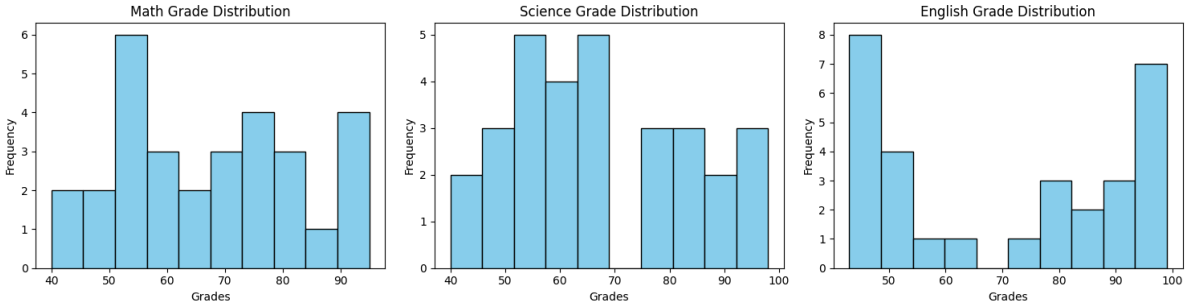
2b) Analyze student grades using numpy

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Create grade data for 30 students (rows) and 3 subjects (columns)
# Columns: [Math, Science, English]
grades = np.random.randint(40, 100, size=(30, 3))

# Step 2: Subject-wise statistics
subjects = ['Math', 'Science', 'English']
subject_avg = np.mean(grades, axis=0)
subject_max = np.max(grades, axis=0)
subject_min = np.min(grades, axis=0)

print("=== Subject-wise Statistics ===")
for i, subject in enumerate(subjects):
    print(f"{subject}: Avg = {subject_avg[i]:.2f}, Max = {subject_max[i]}, Min =

# Step 3: Overall statistics
overall_avg = np.mean(grades)
overall_max = np.max(grades)
overall_min = np.min(grades)

print("\n=== Overall Statistics ===")
print(f"Average Grade: {overall_avg:.2f}")
print(f"Highest Grade: {overall_max}")
print(f"Lowest Grade: {overall_min}")

# Step 4: Student total and average
student_totals = np.sum(grades, axis=1)
student_averages = np.mean(grades, axis=1)

# Step 5: Identify top-performing student(s)
top_score = np.max(student_totals)
top_students = np.where(student_totals == top_score)[0]

print("\n=== Top Performing Students ===")
for i in top_students:
    print(f"Student {i+1}: Total = {student_totals[i]}, Average = {student_averi

# Step 6: Optional - Visualize grade distributions
plt.figure(figsize=(15, 4))

for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.hist(grades[:, i], bins=10, color='skyblue', edgecolor='black')
    plt.title(f'{subjects[i]} Grade Distribution')
    plt.xlabel('Grades')
    plt.ylabel('Frequency')
```

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

```
=== Subject-wise Statistics ===
Math: Avg = 67.93, Max = 95, Min = 40
Science: Avg = 67.63, Max = 98, Min = 40
English: Avg = 70.57, Max = 99, Min = 43

=== Overall Statistics ===
Average Grade: 68.71
Highest Grade: 99
Lowest Grade: 40

=== Top Performing Students ===
Student 17: Total = 277, Average = 92.33
```



3a)Compute total impedance of an electrical circuit

In [5]:

```
import numpy as np

# Frequency input
frequency = float(input("Enter frequency (Hz): "))

# Number of components
n = int(input("Enter number of components: "))

# Initialize empty list for complex impedances
impedances = []

# Get circuit type
config = input("Enter configuration (series/parallel): ").strip().lower()

# Input components
for i in range(n):
    print(f"\nComponent {i+1}:")
    ctype = input("Type (R for resistor, C for capacitor, L for inductor): ").s

    if ctype == 'R':
        R = float(input("Enter resistance (Ohms): "))
        Z = np.complex128(R)
    elif ctype == 'C':
        C = float(input("Enter capacitance (Farads): "))
        Z = 1 / (1j * 2 * np.pi * frequency * C) if C != 0 else np.complex128(n
    elif ctype == 'L':
        L = float(input("Enter inductance (Henrys): "))
```

```
            Z = 1j * 2 * np.pi * frequency * L
        else:
            print("Invalid component type. Skipping.")
            continue

        impedances.append(Z)
        print(f"Impedance of component {i+1}: {Z:.2f} Ω")

    # Convert list to NumPy array
    impedances = np.array(impedances, dtype=np.complex128)

    # Total impedance calculation
    if config == 'series':
        total_impedance = np.sum(impedances)
    elif config == 'parallel':
        total_impedance = 1 / np.sum(1 / impedances)
    else:
        print("Invalid configuration.")
        total_impedance = None

    # Display result
    if total_impedance is not None:
        print(f"\n=== Total Impedance of the Circuit ===")
        print(f"Total Impedance: {total_impedance:.2f} Ω")
        print(f"Magnitude: {np.abs(total_impedance):.2f} Ω")
```

```
Enter frequency (Hz): 50
Enter number of components: 3
Enter configuration (series/parallel): series

Component 1:
Type (R for resistor, C for capacitor, L for inductor): R
Enter resistance (Ohms): 250
Impedance of component 1: 250.00+0.00j Ω

Component 2:
Type (R for resistor, C for capacitor, L for inductor): C
Enter capacitance (Farads): 1
Impedance of component 2: 0.00-0.00j Ω

Component 3:
Type (R for resistor, C for capacitor, L for inductor): L
Enter inductance (Henrys): 0.3
Impedance of component 3: 0.00+94.25j Ω

=== Total Impedance of the Circuit ===
Total Impedance: 250.00+94.24j Ω
Magnitude: 267.17 Ω
```

3b)Perform matrix manipulation on stock price data

In [6]:

```
import numpy as np

# Step 1: Sample data for Stock A and Stock B (5 days, 3 features: Open, High, L
stock_A = np.array([
    [100, 105, 98],
```

```
    [102, 108, 101],
    [104, 110, 103],
    [106, 112, 105],
    [108, 115, 107]
])

stock_B = np.array([
    [99, 104, 97],
    [101, 107, 100],
    [103, 109, 102],
    [105, 111, 104],
    [107, 114, 106]
])

# Step 2: Element-wise sum
sum_matrix = np.add(stock_A, stock_B)

# Step 3: Element-wise difference
diff_matrix = np.subtract(stock_A, stock_B)

# Step 4: Output
print("=== Stock A Price Data ===")
print(stock_A)

print("\n=== Stock B Price Data ===")
print(stock_B)

print("\n=== Element-wise Sum (A + B) ===")
print(sum_matrix)

print("\n=== Element-wise Difference (A - B) ===")
print(diff_matrix)
```

```
=== Stock A Price Data ===
[[100 105  98]
 [102 108 101]
 [104 110 103]
 [106 112 105]
 [108 115 107]]

=== Stock B Price Data ===
[[ 99 104  97]
 [101 107 100]
 [103 109 102]
 [105 111 104]
 [107 114 106]]

=== Element-wise Sum (A + B) ===
[[199 209 195]
 [203 215 201]
 [207 219 205]
 [211 223 209]
 [215 229 213]]

=== Element-wise Difference (A - B) ===
[[1 1 1]
 [1 1 1]
```

```
[1 1 1]
[1 1 1]
[1 1 1]]
```

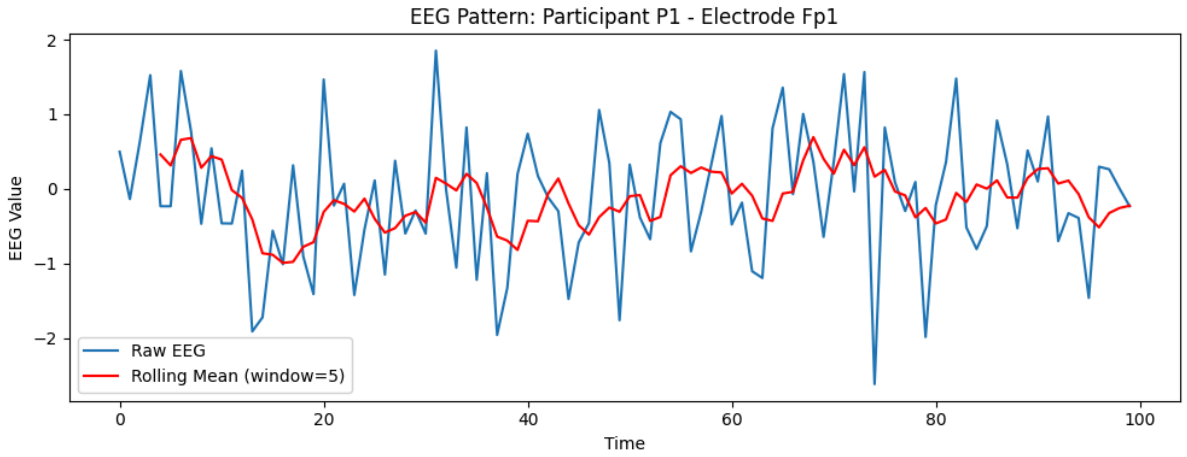
4a)Analyze EEG Data using pandas

In [8]:

```
import pandas as pd
import numpy as np

# Simulate EEG data: 3 participants, 4 electrodes, 100 time points
np.random.seed(42)  # For reproducibility
participants = ['P1', 'P2', 'P3']
electrodes = ['Fp1', 'Fp2', 'Cz', 'Pz']
time_points = 100

# Step 1: Create simulated data
data = []

for participant in participants:
    for electrode in electrodes:
        signal = np.random.normal(loc=0, scale=1, size=time_points)  # Simulate
        for t, value in enumerate(signal):
            data.append([participant, electrode, t, value])

# Step 2: Load into DataFrame
df = pd.DataFrame(data, columns=['Participant', 'Electrode', 'Time', 'EEG_Value

# Step 3: Basic statistics per participant & electrode
stats = df.groupby(['Participant', 'Electrode'])['EEG_Value'].agg(['mean', 'var

# Step 4: Display stats
print("=== Basic EEG Statistics per Participant & Electrode ===")
print(stats)

# Step 5: Optional - Identify patterns/trends using rolling average (example for
df_trend = df[(df['Participant'] == 'P1') & (df['Electrode'] == 'Fp1')].copy()
df_trend['Rolling_Mean'] = df_trend['EEG_Value'].rolling(window=5).mean()

# Optional: Visualization
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))
plt.plot(df_trend['Time'], df_trend['EEG_Value'], label='Raw EEG')
plt.plot(df_trend['Time'], df_trend['Rolling_Mean'], label='Rolling Mean (windo
plt.title('EEG Pattern: Participant P1 - Electrode Fp1')
plt.xlabel('Time')
plt.ylabel('EEG Value')
plt.legend()
plt.tight_layout()
plt.show()
```

```
=== Basic EEG Statistics per Participant & Electrode ===
   Participant Electrode      mean       var       max       min
0           P1        Cz  0.064896  1.175669  3.852731 -3.241267
1           P1       Fp1 -0.103847  0.824770  1.852278 -2.619745
```

```
2            P1       Fp2  0.022305  0.909484  2.720169 -1.918771
3            P1        Pz  0.106840  0.781635  2.189803 -2.123896
4            P2        Cz  0.024094  1.138966  2.573360 -2.696887
5            P2       Fp1 -0.056004  1.131522  3.078881 -2.301921
6            P2       Fp2 -0.115306  0.852922  2.270693 -2.471645
7            P2        Pz -0.006161  0.950136  2.632382 -2.081929
8            P3        Cz  0.160174  0.902951  2.142270 -2.848543
9            P3       Fp1  0.228649  0.878097  2.526932 -1.515744
10           P3       Fp2  0.027854  0.938387  2.163255 -2.423879
11           P3        Pz  0.107092  1.229476  2.439752 -2.896255
```



4b)Perform basic data analysis on product inventory using pandas

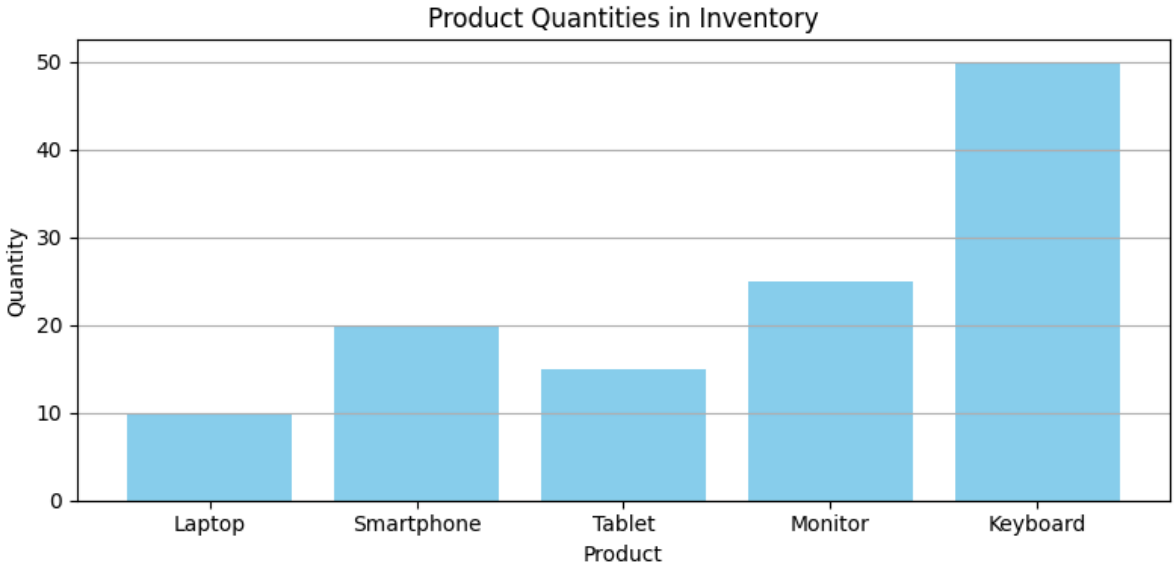
In [9]:

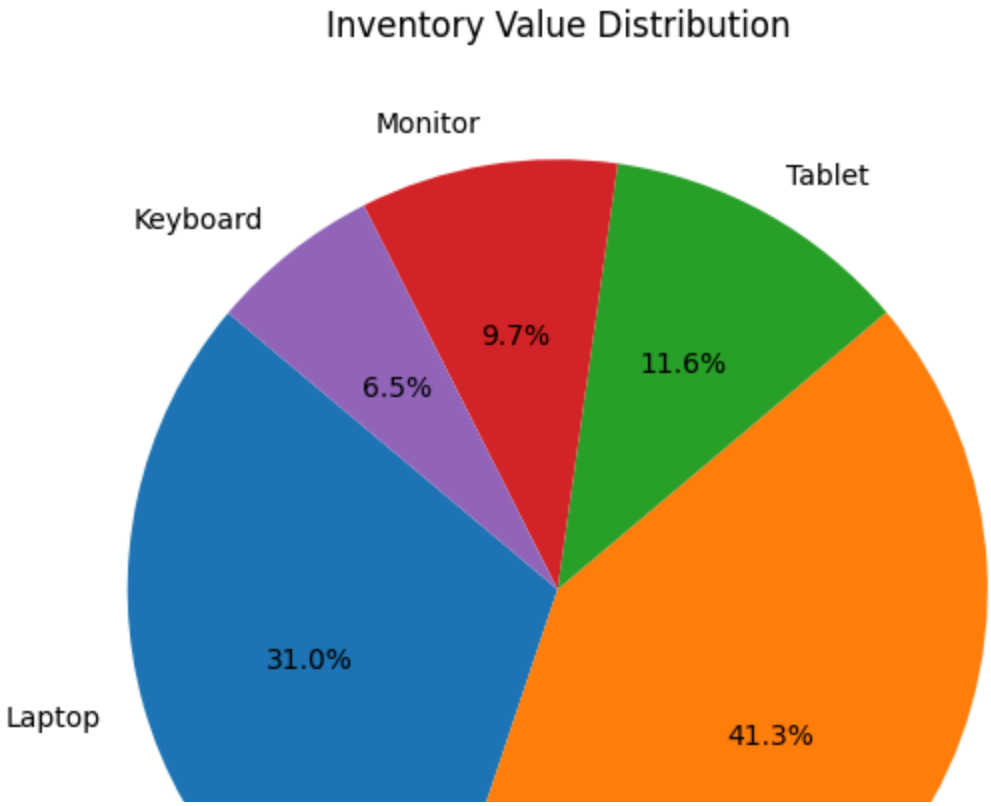
```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Input data
data = {
    'Product': ['Laptop', 'Smartphone', 'Tablet', 'Monitor', 'Keyboard'],
    'Price': [1200, 800, 300, 150, 50],
    'Quantity': [10, 20, 15, 25, 50]
}

# Step 2: Create DataFrame
df = pd.DataFrame(data)

# Step 3: Basic Operations

# 1. Total value per product
df['Total_Value'] = df['Price'] * df['Quantity']

# 2. Total inventory worth
total_inventory_value = df['Total_Value'].sum()

# 3. Most and least expensive product
most_expensive = df.loc[df['Price'].idxmax()]
least_expensive = df.loc[df['Price'].idxmin()]

# 4. Sort by quantity
sorted_by_quantity = df.sort_values(by='Quantity', ascending=False)

# 5. Filter products with quantity > 15
filtered_products = df[df['Quantity'] > 15]
```

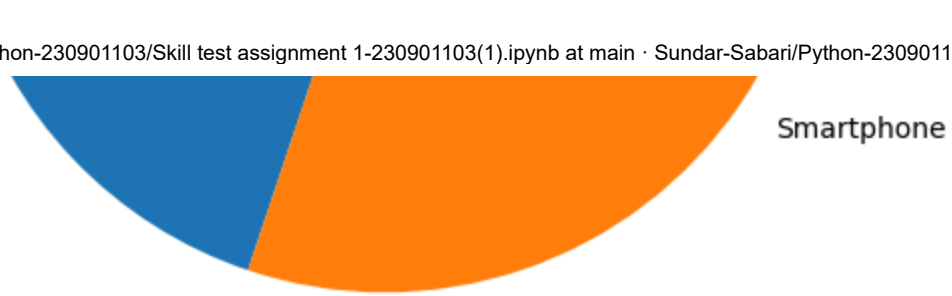
```
# Step 4: Display outputs
print("=== Inventory Data ===")
print(df)

print("\nTotal Inventory Worth: $", total_inventory_value)

print("\nMost Expensive Product:")
print(most_expensive)

print("\nLeast Expensive Product:")
print(least_expensive)

print("\nProducts Sorted by Quantity:")
print(sorted_by_quantity)

print("\nProducts with Quantity > 15:")
print(filtered_products)

# Step 5: Plotting
# Bar chart: Quantity per product
plt.figure(figsize=(8, 4))
plt.bar(df['Product'], df['Quantity'], color='skyblue')
plt.title('Product Quantities in Inventory')
plt.xlabel('Product')
plt.ylabel('Quantity')
plt.grid(axis='y')
plt.tight_layout()
plt.show()

# Pie chart: Share of total inventory value
plt.figure(figsize=(6, 6))
plt.pie(df['Total_Value'], labels=df['Product'], autopct='%1.1f%%', startangle=
plt.title('Inventory Value Distribution')
plt.tight_layout()
plt.show()
```

```
=== Inventory Data ===
      Product  Price  Quantity  Total_Value
0      Laptop   1200        10        12000
1  Smartphone    800        20        16000
2      Tablet    300        15         4500
3     Monitor    150        25         3750
4    Keyboard     50        50         2500

Total Inventory Worth: $ 38750

Most Expensive Product:
Product        Laptop
Price            1200
Quantity           10
Total_Value     12000
Name: 0, dtype: object

Least Expensive Product:
Product        Keyboard
Price                50
```

```
Quantity             50
Total_Value        2500
Name: 4, dtype: object

Products Sorted by Quantity:
      Product  Price  Quantity  Total_Value
4    Keyboard     50        50         2500
3     Monitor    150        25         3750
1  Smartphone    800        20        16000
2      Tablet    300        15         4500
0      Laptop   1200        10        12000

Products with Quantity > 15:
      Product  Price  Quantity  Total_Value
1  Smartphone    800        20        16000
3     Monitor    150        25         3750
4    Keyboard     50        50         2500
```





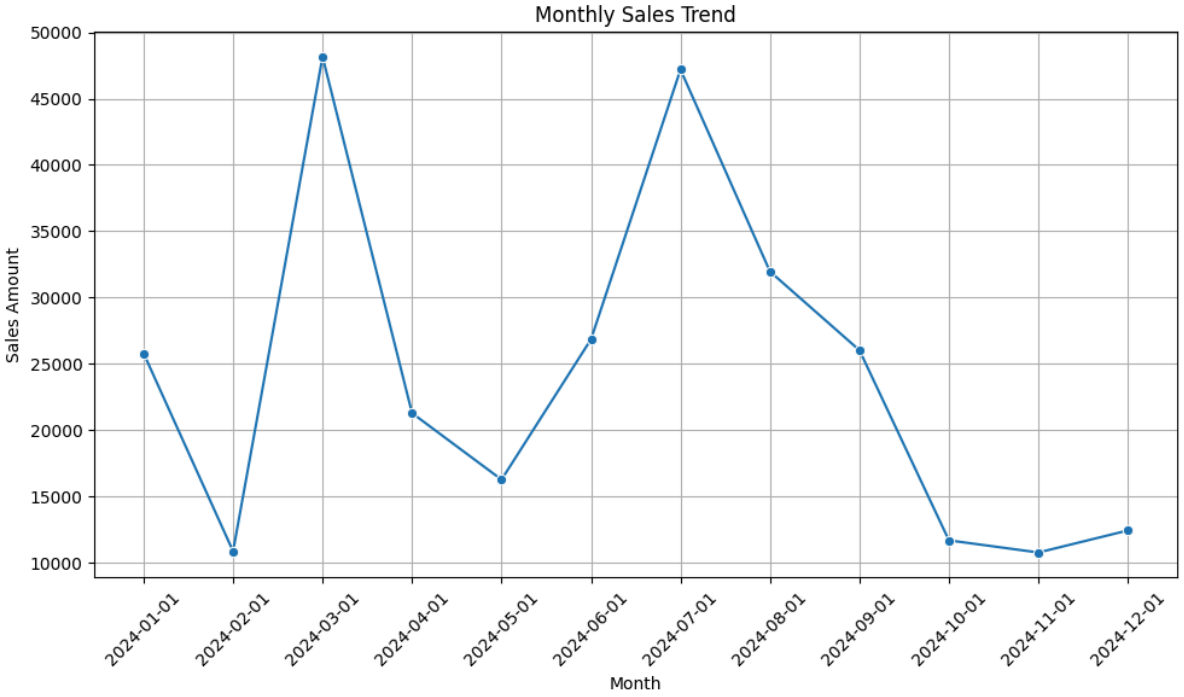


5a)Perform sales data analysis using pandas

In [10]:

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

# Step 1: Generate Random Sales Data
np.random.seed(42)
months = pd.date_range(start='2024-01-01', periods=12, freq='MS')
sales = np.random.randint(10000, 50000, size=12)

# Create DataFrame
sales_df = pd.DataFrame({
    'Month': months,
    'Sales': sales
})

# Step 2: Save DataFrame to CSV
sales_df.to_csv('sales_data.csv', index=False)

# Step 3: Read the CSV file
df = pd.read_csv('sales_data.csv')

# Step 4: Display first 5 rows
print("First 5 rows of sales data:")
print(df.head())

# Step 5: Check for missing values
print("\nMissing values:")
print(df.isnull().sum())

# Optional: Fill or Drop (if any missing data is found)
df.dropna(inplace=True)  # In this case, we assume clean data

# Step 6: Visualize monthly sales trend
plt.figure(figsize=(10, 6))
sns.lineplot(data=df, x='Month', y='Sales', marker='o')
plt.title('Monthly Sales Trend')
plt.xlabel('Month')
plt.ylabel('Sales Amount')
plt.xticks(rotation=45)
plt.tight_layout()
plt.grid(True)
plt.show()
```

```
First 5 rows of sales data:
        Month  Sales
0  2024-01-01  25795
1  2024-02-01  10860
2  2024-03-01  48158
3  2024-04-01  21284
4  2024-05-01  16265

Missing values:
Month    0
Sales    0
dtype: int64
```



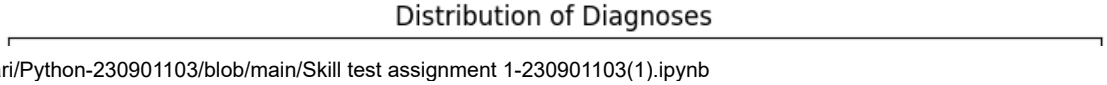
5b)Analyse health care data using python and pandas

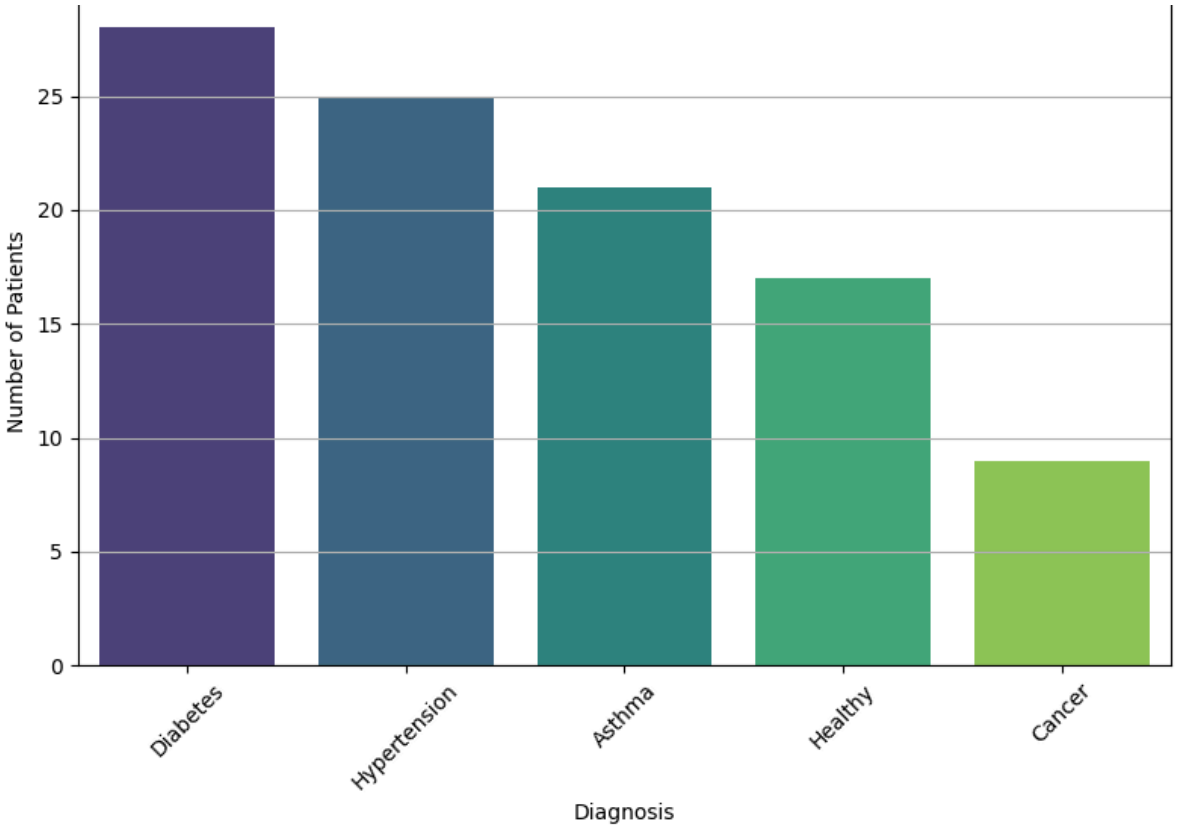
In [11]:

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

# Step 1: Generate Dummy Healthcare Data
np.random.seed(101)
num_records = 100

# Randomly generate data
names = [f"Patient_{i}" for i in range(1, num_records + 1)]
ages = np.random.randint(20, 80, size=num_records)
genders = np.random.choice(['Male', 'Female'], size=num_records)
diagnoses = np.random.choice(['Diabetes', 'Hypertension', 'Asthma', 'Cancer', 'I

# Create DataFrame
healthcare_df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Gender': genders,
    'Diagnosis': diagnoses
```

```
})

# Step 2: Save to CSV
healthcare_df.to_csv('healthcare_data.csv', index=False)

# Step 3: Load the dataset
df = pd.read_csv('healthcare_data.csv')

# Display column names and data types
print("Column Names and Data Types:")
print(df.dtypes)

# Step 4: Calculate average age
average_age = df['Age'].mean()
print(f"\nAverage Age of Patients: {average_age:.2f} years")

# Step 5: Analyze distribution of diagnoses
diagnosis_counts = df['Diagnosis'].value_counts()
print("\nDiagnosis Distribution:")
print(diagnosis_counts)

# Step 6: Visualize the diagnosis distribution
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='Diagnosis', order=diagnosis_counts.index, palette='vi
plt.title('Distribution of Diagnoses')
plt.xlabel('Diagnosis')
plt.ylabel('Number of Patients')
plt.xticks(rotation=45)
plt.tight_layout()
plt.grid(True, axis='y')
plt.show()
```

```
Column Names and Data Types:
Name         object
Age           int64
Gender       object
Diagnosis    object
dtype: object

Average Age of Patients: 49.91 years

Diagnosis Distribution:
Diagnosis
Diabetes        28
Hypertension    25
Asthma          21
Healthy         17
Cancer           9
Name: count, dtype: int64
<ipython-input-11-792daebb274b>:45: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.
14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data=df, x='Diagnosis', order=diagnosis_counts.index, palette='vir
idis')
```




Number of Patients
0
5
10
15
20
25
Diabetes
Hypertension
Asthma
Healthy
Cancer
Diagnosis