📊 Complete Data Science Practice: NumPy, Pandas & Matplotlib

1. NumPy Fundamentals

In [1]:
```
# 📦 Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

print("✅ Libraries imported successfully!")
```

```
✅ Libraries imported successfully!
```

In [2]:
```
# 🎯 1. Array Creation & Basic Operations
print("🎯 1. NUMPY ARRAY BASICS")
print("="*50)

# Create arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.arange(1, 11)  # 1 to 10
arr3 = np.linspace(0, 1, 5)  # 5 equally spaced values
zeros = np.zeros((3, 3))
ones = np.ones((2, 4))
identity = np.eye(4)
random_arr = np.random.rand(5)

print(f"Simple array: {arr1}")
print(f"arange: {arr2}")
print(f"linspace: {arr3}")
print(f"\nZeros matrix:\n{zeros}")
print(f"\nOnes matrix:\n{ones}")
print(f"\nIdentity matrix:\n{identity}")
print(f"\nRandom array: {random_arr}")
```

```
🎯 1. NUMPY ARRAY BASICS
==================================================
Simple array: [1 2 3 4 5]
arange: [ 1  2  3  4  5  6  7  8  9 10]
linspace: [0.   0.25 0.5  0.75 1.  ]

Zeros matrix:
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]

Ones matrix:
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]

Identity matrix:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Random array: [0.54327993 0.63467729 0.68924604 0.69511724 0.67940271]
```

In [3]:

```python
# 🔢 2. Array Mathematics
print("\n🔢 2. ARRAY MATHEMATICS")
print("="*50)

a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])

print(f"Array a: {a}")
print(f"Array b: {b}")
print(f"\nAddition: {a + b}")
print(f"Subtraction: {b - a}")
print(f"Multiplication: {a * b}")
print(f"Division: {b / a}")
print(f"\nSquare of a: {a ** 2}")
print(f"Square root of b: {np.sqrt(b)}")
print(f"Exponential of a: {np.exp(a)}")
```

```
🔢 2. ARRAY MATHEMATICS
==================================================
Array a: [1 2 3 4]
Array b: [5 6 7 8]

Addition: [ 6  8 10 12]
Subtraction: [4 4 4 4]
Multiplication: [ 5 12 21 32]
Division: [5.         3.         2.33333333 2.        ]

Square of a: [ 1  4  9 16]
Square root of b: [2.23606798 2.44948974 2.64575131 2.82842712]
Exponential of a: [ 2.71828183  7.3890561  20.08553692 54.59815003]
```

In [4]:

```python
# 📐 3. Multi-dimensional Arrays
print("\n📐 3. MULTI-DIMENSIONAL ARRAYS")
print("="*50)

matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
```

```
print(f"2D Matrix:\n{matrix}")
print(f"\nShape: {matrix.shape}")
print(f"Dimensions: {matrix.ndim}")
print(f"Size: {matrix.size}")
print(f"Data type: {matrix.dtype}")
print(f"\nFirst row: {matrix[0]}")
print(f"Second column: {matrix[:, 1]}")
print(f"Element at [1,2]: {matrix[1, 2]}")
```

```
📐 3. MULTI-DIMENSIONAL ARRAYS
==================================================
2D Matrix:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Shape: (3, 3)
Dimensions: 2
Size: 9
Data type: int64

First row: [1 2 3]
Second column: [2 5 8]
Element at [1,2]: 6
```

In [5]:

```
# 🎲 4. Array Manipulation
print("\n🎲 4. ARRAY MANIPULATION")
print("="*50)

arr = np.array([1, 2, 3, 4, 5, 6])

print(f"Original array: {arr}")
print(f"\nReshaped to 2x3:\n{arr.reshape(2, 3)}")
print(f"\nTranspose of 2x3 matrix:\n{arr.reshape(2, 3).T}")
print(f"\nFlattened array: {arr.reshape(2, 3).flatten()}")
print(f"\nSorted array: {np.sort(np.array([3, 1, 4, 1, 5]))}")
print(f"Reversed array: {arr[::-1]}")
```

```
🎲 4. ARRAY MANIPULATION
==================================================
Original array: [1 2 3 4 5 6]

Reshaped to 2x3:
[[1 2 3]
 [4 5 6]]

Transpose of 2x3 matrix:
[[1 4]
 [2 5]
 [3 6]]

Flattened array: [1 2 3 4 5 6]

Sorted array: [1 1 3 4 5]
Reversed array: [6 5 4 3 2 1]
```

In [6]:

```
# 📊 5. Statistical Operations
print("\n📊 5. STATISTICAL OPERATIONS")
print("="*50)

data = np.array([23, 45, 67, 89, 34, 56, 78, 90, 12, 45])
```

```
print(f"Dataset: {data}")
print(f"\nMean: {np.mean(data)}")
print(f"Median: {np.median(data)}")
print(f"Standard Deviation: {np.std(data)}")
print(f"Variance: {np.var(data)}")
print(f"Minimum: {np.min(data)}")
print(f"Maximum: {np.max(data)}")
print(f"Sum: {np.sum(data)}")
print(f"Product: {np.prod(data[:5])}")
```

```
📊 5. STATISTICAL OPERATIONS
==================================================
Dataset: [23 45 67 89 34 56 78 90 12 45]

Mean: 53.9
Median: 50.5
Standard Deviation: 25.606444501335986
Variance: 655.6899999999999
Minimum: 12
Maximum: 90
Sum: 539
Product: 209837970
```

2. Pandas Data Handling

In [7]:

```
# 📋 6. DataFrame Creation
print("📋 6. DATAFRAME CREATION")
print("="*50)

# Create DataFrame from dictionary
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'Diana', 'Eve'],
    'Age': [25, 30, 35, 28, 32],
    'City': ['NYC', 'LA', 'Chicago', 'Miami', 'Seattle'],
    'Salary': [50000, 60000, 75000, 55000, 80000],
    'Department': ['HR', 'IT', 'Finance', 'Marketing', 'IT']
}

df = pd.DataFrame(data)
print("DataFrame created successfully!")
print(f"\nShape: {df.shape}")
print(f"\nFirst 3 rows:")
print(df.head(3))
print(f"\nDataFrame Info:")
print(df.info())
```

```
📋 6. DATAFRAME CREATION
==================================================
DataFrame created successfully!

Shape: (5, 5)

First 3 rows:
      Name  Age     City  Salary Department
0    Alice   25      NYC   50000         HR
1      Bob   30       LA   60000         IT
2  Charlie   35  Chicago   75000    Finance

DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Name        5 non-null      object
 1   Age         5 non-null      int64
 2   City        5 non-null      object
 3   Salary      5 non-null      int64
 4   Department  5 non-null      object
dtypes: int64(2), object(3)
memory usage: 332.0+ bytes
None
```

In [8]:

```python
# 🔍 7. Data Selection & Filtering
print("\n🔍 7. DATA SELECTION & FILTERING")
print("="*50)

print("Select specific columns:")
print(df[['Name', 'Salary']])

print("\nSelect rows by index:")
print(df.iloc[1:4])  # Rows 1, 2, 3

print("\nFilter: Age > 30:")
print(df[df['Age'] > 30])

print("\nMultiple conditions (Age > 28 AND Department == 'IT'):")
print(df[(df['Age'] > 28) & (df['Department'] == 'IT')])

print("\nFilter by string contains:")
print(df[df['Name'].str.contains('li')])
```

```
🔍 7. DATA SELECTION & FILTERING
==================================================
Select specific columns:
      Name  Salary
0    Alice   50000
1      Bob   60000
2  Charlie   75000
3    Diana   55000
4      Eve   80000

Select rows by index:
      Name  Age     City  Salary Department
1      Bob   30       LA   60000         IT
2  Charlie   35  Chicago   75000    Finance
3    Diana   28    Miami   55000  Marketing

Filter: Age > 30:
      Name  Age     City  Salary Department
2  Charlie   35  Chicago   75000    Finance
4      Eve   32  Seattle   80000         IT

Multiple conditions (Age > 28 AND Department == 'IT'):
  Name  Age     City  Salary Department
1  Bob   30       LA   60000         IT
4  Eve   32  Seattle   80000         IT

Filter by string contains:
      Name  Age     City  Salary Department
0    Alice   25      NYC   50000         HR
2  Charlie   35  Chicago   75000    Finance
```

In [9]:

```python
# 📈 8. Data Manipulation
print("\n📈 8. DATA MANIPULATION")
print("="*50)

# Add new column
df['Bonus'] = df['Salary'] * 0.1
df['Total_Compensation'] = df['Salary'] + df['Bonus']

# Modify existing column
df['Age_Group'] = pd.cut(df['Age'],
                         bins=[20, 30, 40, 50],
                         labels=['20s', '30s', '40s'])

# Drop column
df_dropped = df.drop('City', axis=1)

print("DataFrame with new columns:")
print(df)

print("\nSorted by Salary (descending):")
print(df.sort_values('Salary', ascending=False))
```

```
📈 8. DATA MANIPULATION
==================================================
DataFrame with new columns:
      Name  Age     City  Salary Department   Bonus  Total_Compensation  \
0    Alice   25      NYC   50000         HR  5000.0             55000.0
1      Bob   30       LA   60000         IT  6000.0             66000.0
2  Charlie   35  Chicago   75000    Finance  7500.0             82500.0
3    Diana   28    Miami   55000  Marketing  5500.0             60500.0
4      Eve   32  Seattle   80000         IT  8000.0             88000.0

  Age_Group
0       20s
1       20s
2       30s
3       20s
4       30s

Sorted by Salary (descending):
      Name  Age     City  Salary Department   Bonus  Total_Compensation  \
4      Eve   32  Seattle   80000         IT  8000.0             88000.0
2  Charlie   35  Chicago   75000    Finance  7500.0             82500.0
1      Bob   30       LA   60000         IT  6000.0             66000.0
3    Diana   28    Miami   55000  Marketing  5500.0             60500.0
0    Alice   25      NYC   50000         HR  5000.0             55000.0

  Age_Group
4       30s
2       30s
1       20s
3       20s
0       20s
```

In [ ]:

```python
# 📊 9. Aggregation & Grouping
print("\n📊 9. AGGREGATION & GROUPING")
print("="*50)

# Basic statistics
print("Basic Statistics:")
print(df.describe())

print("\nStatistics for numeric columns:")
print(df[['Age', 'Salary', 'Bonus']].agg(['mean', 'median', 'std', 'sum']))

print("\nGroup by Department:")
grouped = df.groupby('Department').agg({
    'Age': 'mean',
    'Salary': ['mean', 'max', 'min', 'count'],
    'Bonus': 'sum'
})
print(grouped)
```

In [ ]:

```python
# 🎭 10. Handling Missing Data
print("\n🎭 10. HANDLING MISSING DATA")
print("="*50)

# Create DataFrame with missing values
df_missing = df.copy()
df_missing.loc[1, 'Salary'] = np.nan
df_missing.loc[3, 'Age'] = np.nan
df_missing.loc[4, 'Department'] = np.nan
```

```
print("DataFrame with missing values:")
print(df_missing)

print(f"\nMissing values count:\n{df_missing.isnull().sum()}")

# Fill missing values
df_filled = df_missing.copy()
df_filled['Salary'].fillna(df_filled['Salary'].mean(), inplace=True)
df_filled['Age'].fillna(df_filled['Age'].median(), inplace=True)
df_filled['Department'].fillna('Unknown', inplace=True)

print("\nAfter filling missing values:")
print(df_filled)
```

In [10]:

```
# 🔗 11. Merging & Joining DataFrames
print("\n🔗 11. MERGING & JOINING DATAFRAMES")
print("="*50)

# Create additional DataFrames
df1 = pd.DataFrame({
    'ID': [1, 2, 3, 4],
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Dept_ID': [101, 102, 101, 103]
})

df2 = pd.DataFrame({
    'Dept_ID': [101, 102, 103, 104],
    'Department': ['HR', 'IT', 'Finance', 'Marketing'],
    'Location': ['NYC', 'LA', 'Chicago', 'Miami']
})

print("DataFrame 1:")
print(df1)
print("\nDataFrame 2:")
print(df2)

# Merge DataFrames
merged_df = pd.merge(df1, df2, on='Dept_ID', how='inner')
print("\nMerged DataFrame (inner join):")
print(merged_df)

# Concatenate DataFrames
df3 = pd.DataFrame({'ID': [5], 'Name': ['Eve'], 'Dept_ID': [104]})
concatenated = pd.concat([df1, df3], ignore_index=True)
print("\nConcatenated DataFrame:")
print(concatenated)
```

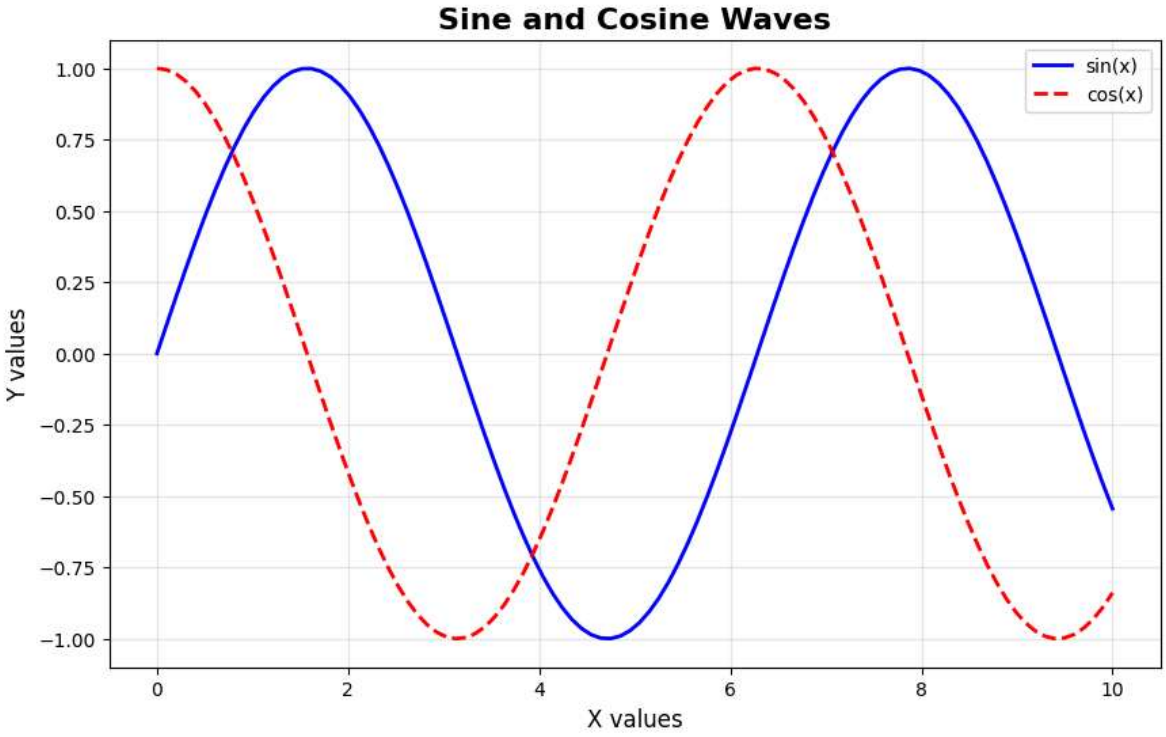
```
🔗 11. MERGING & JOINING DATAFRAMES
==================================================
DataFrame 1:
   ID     Name  Dept_ID
0   1    Alice      101
1   2      Bob      102
2   3  Charlie      101
3   4    David      103

DataFrame 2:
   Dept_ID Department Location
0      101         HR      NYC
1      102         IT       LA
2      103    Finance  Chicago
3      104  Marketing    Miami

Merged DataFrame (inner join):
   ID     Name  Dept_ID Department Location
0   1    Alice      101         HR      NYC
1   2      Bob      102         IT       LA
2   3  Charlie      101         HR      NYC
3   4    David      103    Finance  Chicago

Concatenated DataFrame:
   ID     Name  Dept_ID
0   1    Alice      101
1   2      Bob      102
2   3  Charlie      101
3   4    David      103
4   5      Eve      104
```

3. Matplotlib Visualization

In [11]:

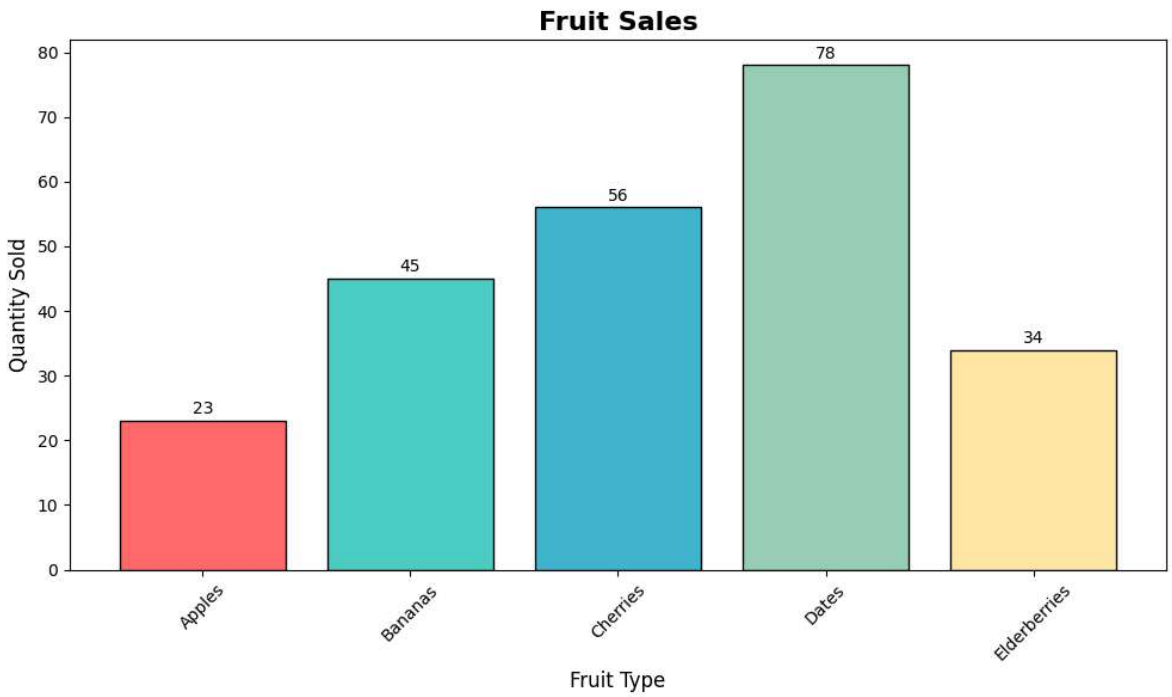
```
# 📊 12. Basic Line Plot
print("📊 12. BASIC LINE PLOT")
print("="*50)

# Create data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='sin(x)', color='blue', linewidth=2, linestyle='-')
plt.plot(x, y2, label='cos(x)', color='red', linewidth=2, linestyle='--')

plt.title('Sine and Cosine Waves', fontsize=16, fontweight='bold')
plt.xlabel('X values', fontsize=12)
plt.ylabel('Y values', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
📊 12. BASIC LINE PLOT
==================================================
```



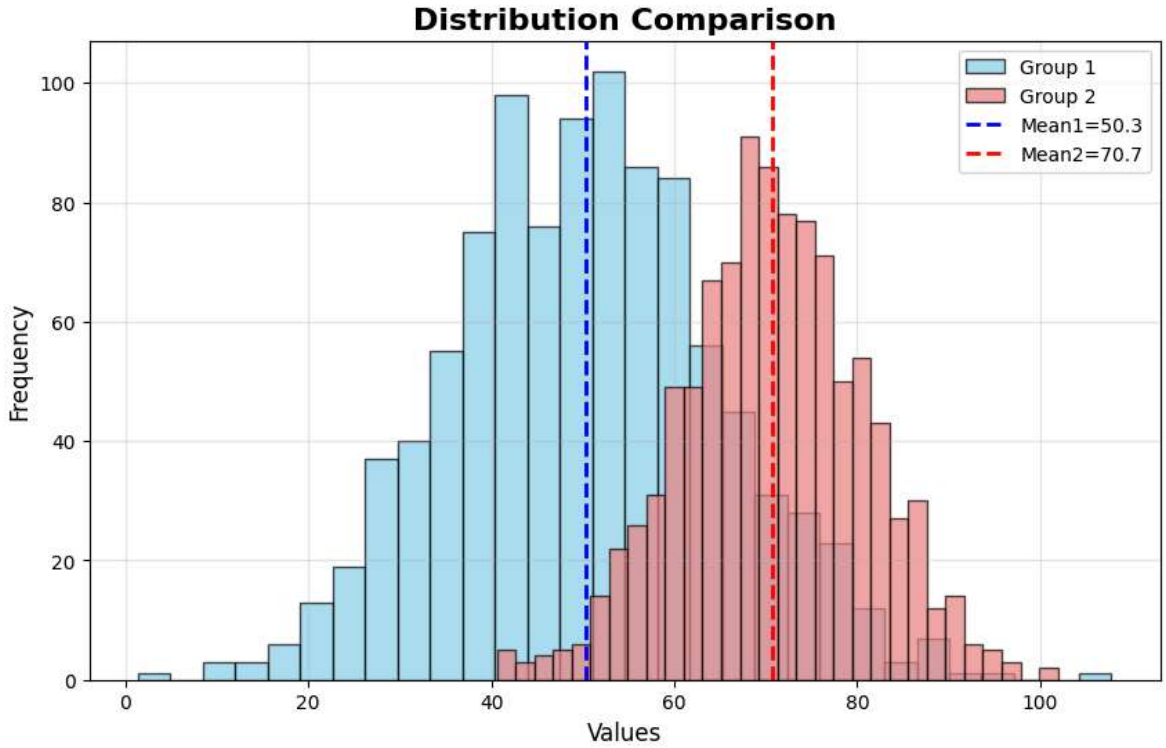
In [12]:
```
# 📊 13. Bar Chart
print("\n📊 13. BAR CHART")
print("="*50)

categories = ['Apples', 'Bananas', 'Cherries', 'Dates', 'Elderberries']
values = [23, 45, 56, 78, 34]
colors = ['#FF6B6B', '#4ECDC4', '#45B7D1', '#96CEB4', '#FFEAA7']

plt.figure(figsize=(10, 6))
bars = plt.bar(categories, values, color=colors, edgecolor='black')

plt.title('Fruit Sales', fontsize=16, fontweight='bold')
plt.xlabel('Fruit Type', fontsize=12)
plt.ylabel('Quantity Sold', fontsize=12)
plt.xticks(rotation=45)

# Add value labels on bars
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height + 0.5,
             f'{int(height)}', ha='center', va='bottom')

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

```
📊 13. BAR CHART
==================================================
```



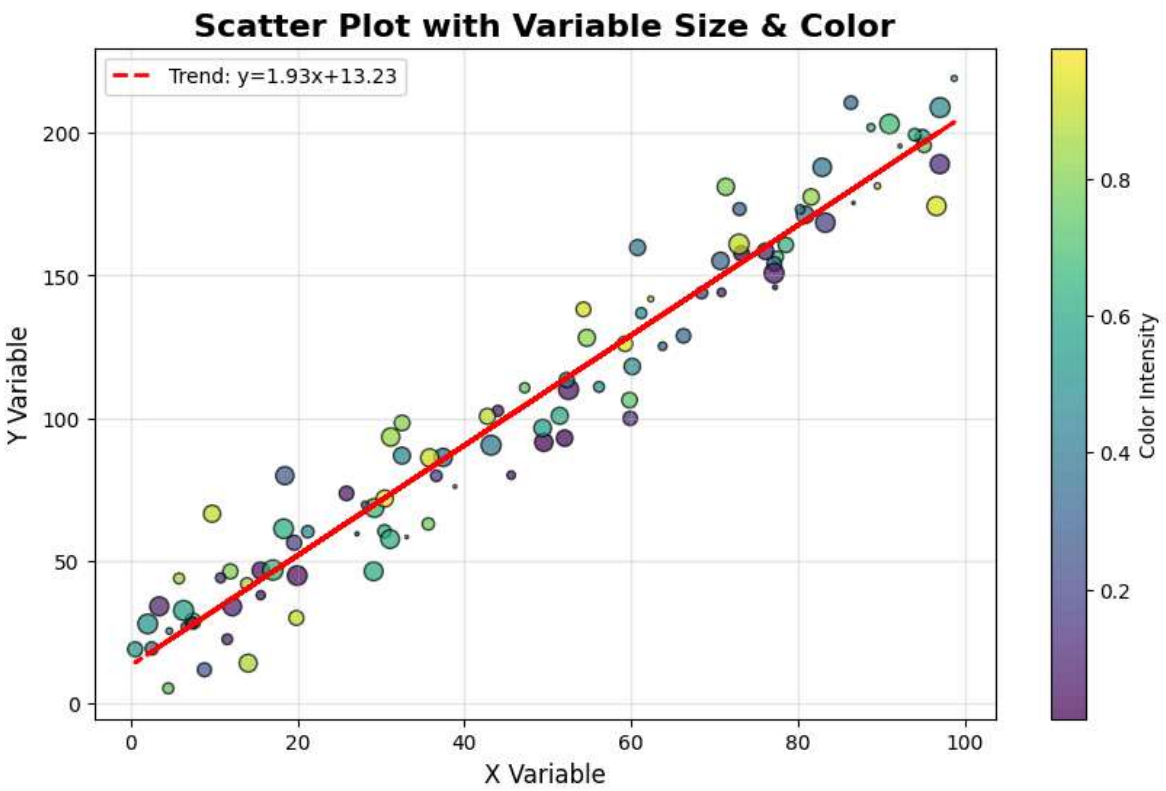
In [13]:
```
# 📊 14. Histogram
print("\n📊 14. HISTOGRAM")
print("="*50)

# Generate random data
np.random.seed(42)
data1 = np.random.normal(50, 15, 1000)  # Mean=50, Std=15
data2 = np.random.normal(70, 10, 1000)  # Mean=70, Std=10

plt.figure(figsize=(10, 6))
plt.hist(data1, bins=30, alpha=0.7, label='Group 1', color='skyblue', edgecolor=
plt.hist(data2, bins=30, alpha=0.7, label='Group 2', color='lightcoral', edgecol

plt.title('Distribution Comparison', fontsize=16, fontweight='bold')
plt.xlabel('Values', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.legend()
plt.grid(True, alpha=0.3)

# Add mean lines
plt.axvline(data1.mean(), color='blue', linestyle='--', linewidth=2, label=f'Mea
plt.axvline(data2.mean(), color='red', linestyle='--', linewidth=2, label=f'Mean

plt.legend()
plt.show()
```

```
📊 14. HISTOGRAM
==================================================
```



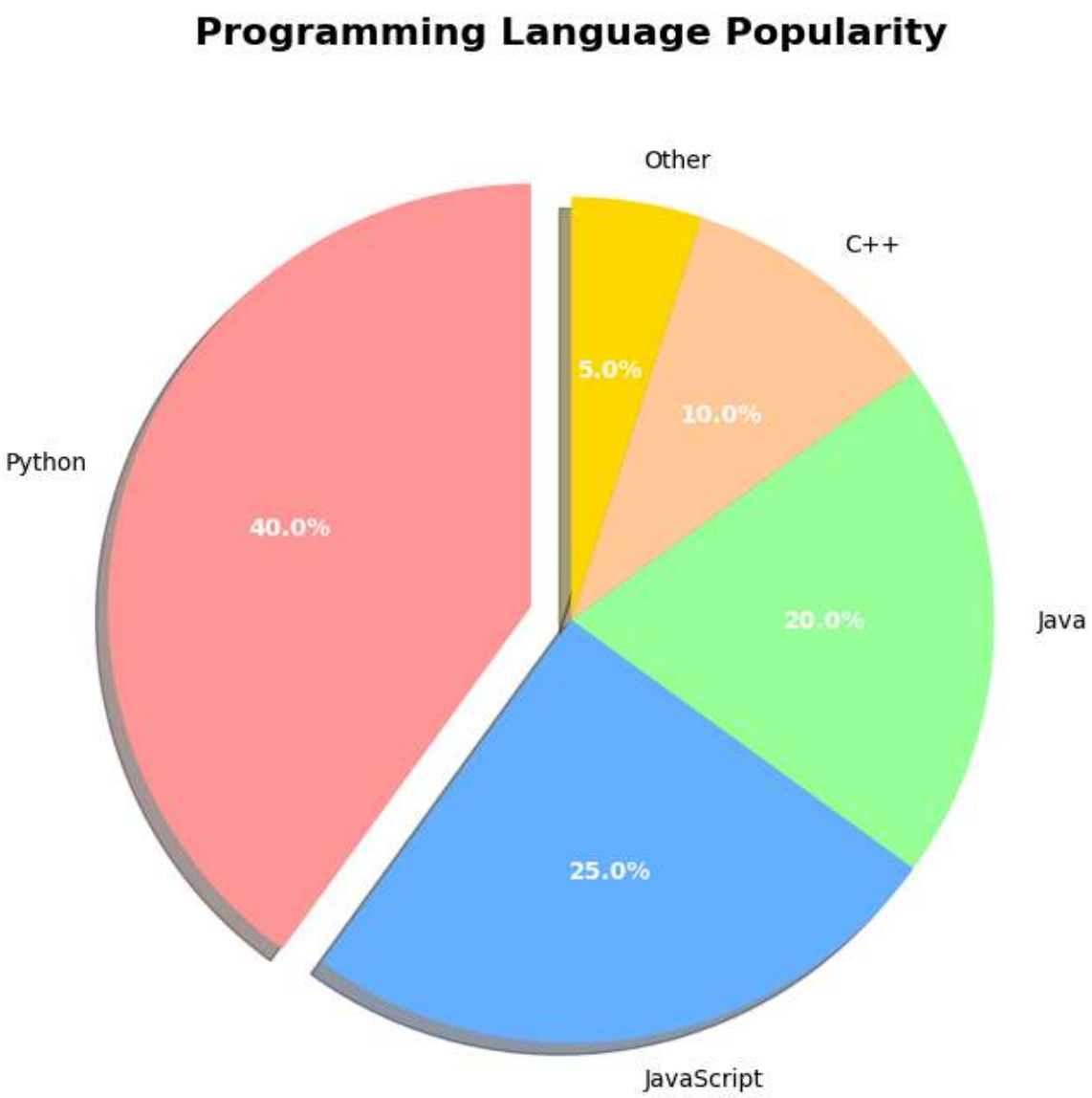
In [14]:
```
# 📊 15. Scatter Plot
print("\n📊 15. SCATTER PLOT")
print("="*50)

# Create sample data
np.random.seed(42)
n_points = 100
x = np.random.rand(n_points) * 100
y = 2 * x + 10 + np.random.randn(n_points) * 15  # Linear relationship with nois
sizes = np.random.rand(n_points) * 100  # Variable point sizes
colors = np.random.rand(n_points)  # Variable colors

plt.figure(figsize=(10, 6))
scatter = plt.scatter(x, y, s=sizes, c=colors, alpha=0.7, cmap='viridis', edgecc

plt.title('Scatter Plot with Variable Size & Color', fontsize=16, fontweight='bo
plt.xlabel('X Variable', fontsize=12)
plt.ylabel('Y Variable', fontsize=12)
plt.colorbar(scatter, label='Color Intensity')

# Add trend line
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, p(x), "r--", linewidth=2, label=f'Trend: y={z[0]:.2f}x+{z[1]:.2f}')

plt.legend()
plt.grid(True, alpha=0.3)
plt.show()
```

```
📊 15. SCATTER PLOT
==================================================
```



In [15]:
```
# 📊 16. Pie Chart
print("\n📊 16. PIE CHART")
print("="*50)

labels = ['Python', 'JavaScript', 'Java', 'C++', 'Other']
sizes = [40, 25, 20, 10, 5]
colors = ['#FF9999', '#66B2FF', '#99FF99', '#FFCC99', '#FFD700']
explode = (0.1, 0, 0, 0, 0)  # Explode Python slice

plt.figure(figsize=(8, 8))
wedges, texts, autotexts = plt.pie(sizes, explode=explode, labels=labels, colors
                                   autopct='%1.1f%%', shadow=True, startangle=90

# Style the text
for autotext in autotexts:
    autotext.set_color('white')
    autotext.set_fontweight('bold')

plt.title('Programming Language Popularity', fontsize=16, fontweight='bold', pad
plt.show()
```

```
📊 16. PIE CHART
==================================================
```

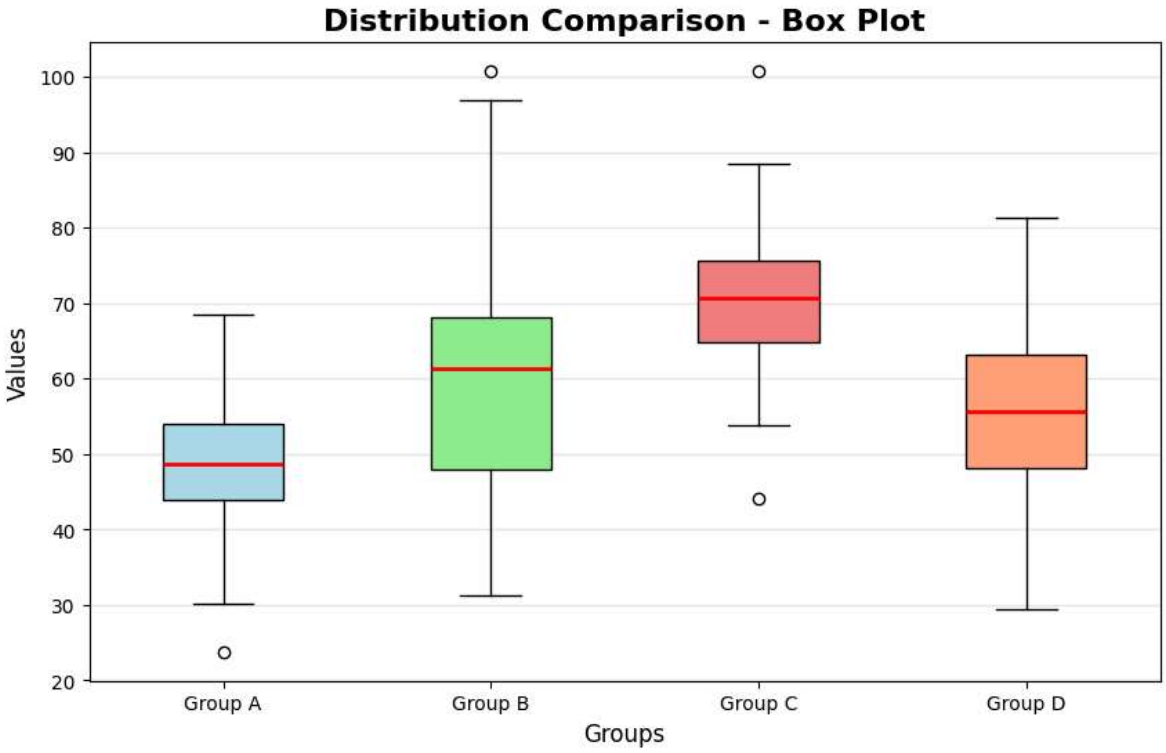


In [16]:
```
# 📊 17. Box Plot
print("\n📊 17. BOX PLOT")
print("="*50)

# Create sample data
np.random.seed(42)
data_groups = [
    np.random.normal(50, 10, 100),
    np.random.normal(60, 15, 100),
    np.random.normal(70, 8, 100),
    np.random.normal(55, 12, 100)
]

labels = ['Group A', 'Group B', 'Group C', 'Group D']

plt.figure(figsize=(10, 6))
box = plt.boxplot(data_groups, labels=labels, patch_artist=True,
                  medianprops={'color': 'red', 'linewidth': 2})

# Customize colors
colors = ['lightblue', 'lightgreen', 'lightcoral', 'lightsalmon']
for patch, color in zip(box['boxes'], colors):
    patch.set_facecolor(color)
```

```
plt.title('Distribution Comparison - Box Plot', fontsize=16, fontweight='bold')
plt.xlabel('Groups', fontsize=12)
plt.ylabel('Values', fontsize=12)
plt.grid(True, alpha=0.3, axis='y')
plt.show()
```

```
📊 17. BOX PLOT
==================================================
```

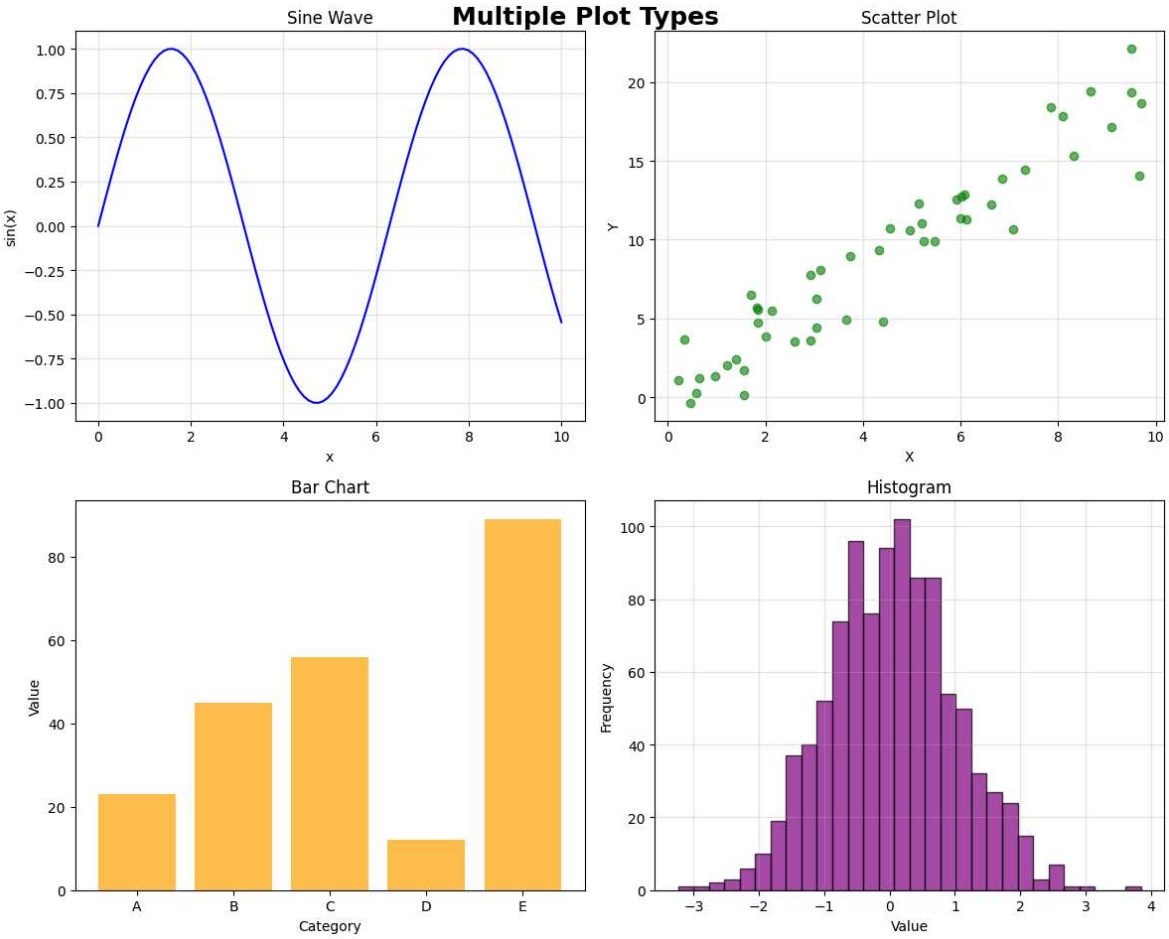
```
C:\Users\mohda\AppData\Local\Temp\ipykernel_2776\3915155315.py:17: MatplotlibDepr
ecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_label
s' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
  box = plt.boxplot(data_groups, labels=labels, patch_artist=True,
```



In [17]:
```
# 📊 18. Subplots (Multiple Plots)
print("\n📊 18. SUBPLOTS (MULTIPLE PLOTS)")
print("="*50)

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
fig.suptitle('Multiple Plot Types', fontsize=18, fontweight='bold', y=0.95)

# Plot 1: Line plot
x = np.linspace(0, 10, 100)
axes[0, 0].plot(x, np.sin(x), 'b-')
axes[0, 0].set_title('Sine Wave')
axes[0, 0].set_xlabel('x')
axes[0, 0].set_ylabel('sin(x)')
axes[0, 0].grid(True, alpha=0.3)

# Plot 2: Scatter plot
np.random.seed(42)
x_scatter = np.random.rand(50) * 10
y_scatter = x_scatter * 2 + np.random.randn(50) * 2
axes[0, 1].scatter(x_scatter, y_scatter, alpha=0.6, c='green')
axes[0, 1].set_title('Scatter Plot')
axes[0, 1].set_xlabel('X')
axes[0, 1].set_ylabel('Y')
```

```
axes[0, 1].grid(True, alpha=0.3)

# Plot 3: Bar plot
categories = ['A', 'B', 'C', 'D', 'E']
values = [23, 45, 56, 12, 89]
axes[1, 0].bar(categories, values, color='orange', alpha=0.7)
axes[1, 0].set_title('Bar Chart')
axes[1, 0].set_xlabel('Category')
axes[1, 0].set_ylabel('Value')

# Plot 4: Histogram
data_hist = np.random.normal(0, 1, 1000)
axes[1, 1].hist(data_hist, bins=30, alpha=0.7, color='purple', edgecolor='black'
axes[1, 1].set_title('Histogram')
axes[1, 1].set_xlabel('Value')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].grid(True, alpha=0.3)

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

```
📊 18. SUBPLOTS (MULTIPLE PLOTS)
==================================================
```

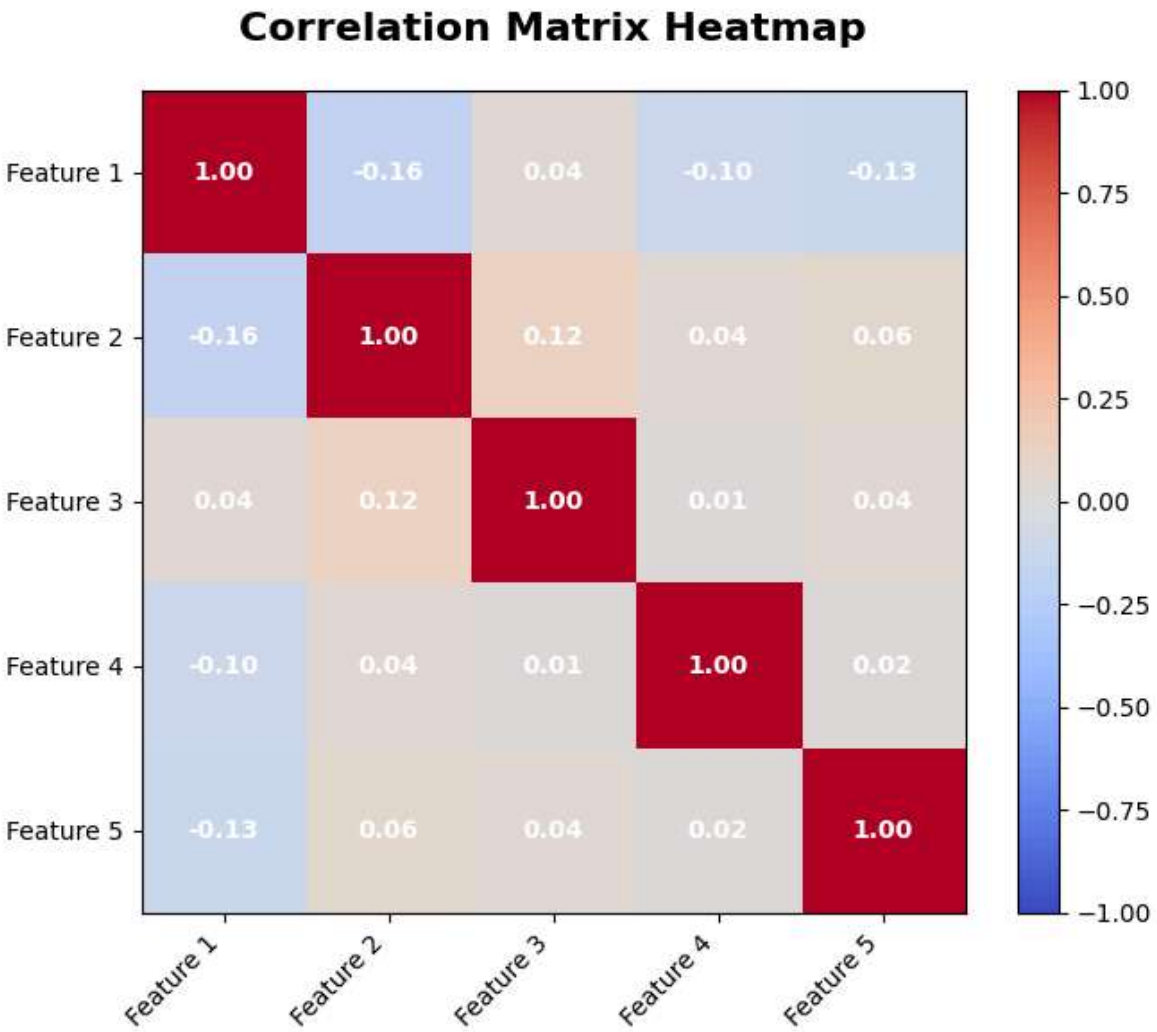


In [18]:

```
# 📊 19. Heatmap
print("\n📊 19. HEATMAP")
print("="*50)

# Create correlation matrix
np.random.seed(42)
data = np.random.randn(100, 5)
corr_matrix = np.corrcoef(data.T)
```

```
labels = ['Feature 1', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5']

plt.figure(figsize=(8, 6))
heatmap = plt.imshow(corr_matrix, cmap='coolwarm', vmin=-1, vmax=1)

# Add colorbar
plt.colorbar(heatmap, fraction=0.046, pad=0.04)

# Add text annotations
for i in range(len(labels)):
    for j in range(len(labels)):
        text = plt.text(j, i, f'{corr_matrix[i, j]:.2f}',
                        ha='center', va='center', color='white', fontweight='bold

plt.title('Correlation Matrix Heatmap', fontsize=16, fontweight='bold', pad=20)
plt.xticks(range(len(labels)), labels, rotation=45, ha='right')
plt.yticks(range(len(labels)), labels)
plt.tight_layout()
plt.show()
```

📊 19. HEATMAP
==========

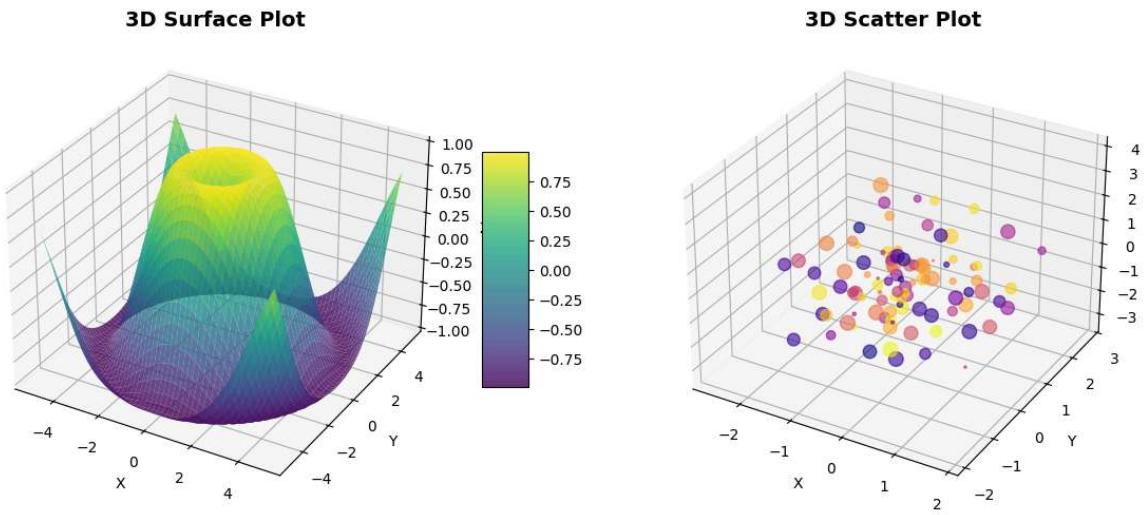


In [19]:

```
# 📊 20. 3D Plot
print("\n📊 20. 3D PLOT")
print("="*50)

from mpl_toolkits.mplot3d import Axes3D
```

```
# Create data for 3D plot
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))

fig = plt.figure(figsize=(12, 5))

# 3D Surface plot
ax1 = fig.add_subplot(121, projection='3d')
surf = ax1.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none', alpha=0.8)
ax1.set_title('3D Surface Plot', fontsize=14, fontweight='bold')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')
fig.colorbar(surf, ax=ax1, shrink=0.5, aspect=5)

# 3D Scatter plot
ax2 = fig.add_subplot(122, projection='3d')
np.random.seed(42)
n_points = 100
x_scatter = np.random.randn(n_points)
y_scatter = np.random.randn(n_points)
z_scatter = np.random.randn(n_points)
colors = np.random.rand(n_points)
sizes = 100 * np.random.rand(n_points)

scatter = ax2.scatter(x_scatter, y_scatter, z_scatter, c=colors, s=sizes, alpha=
ax2.set_title('3D Scatter Plot', fontsize=14, fontweight='bold')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')

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

📊 20. 3D PLOT
==========



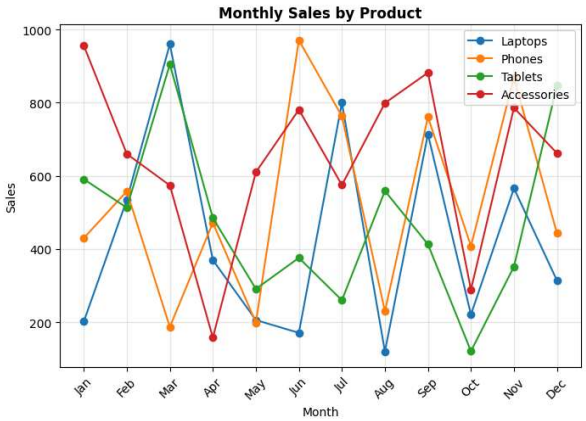
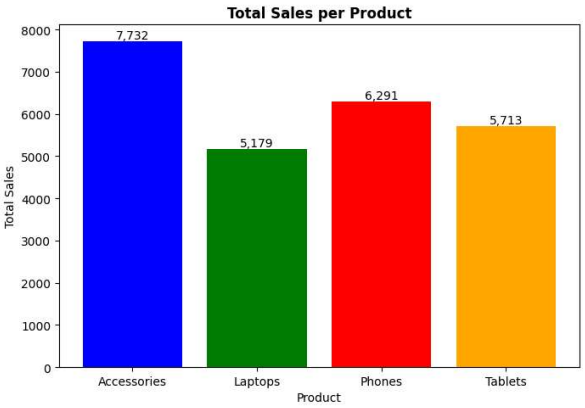
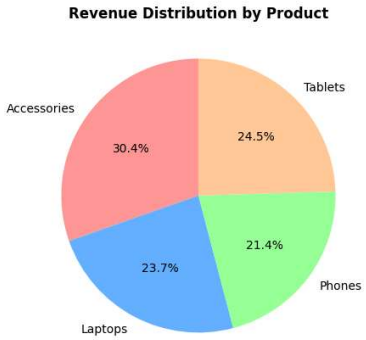
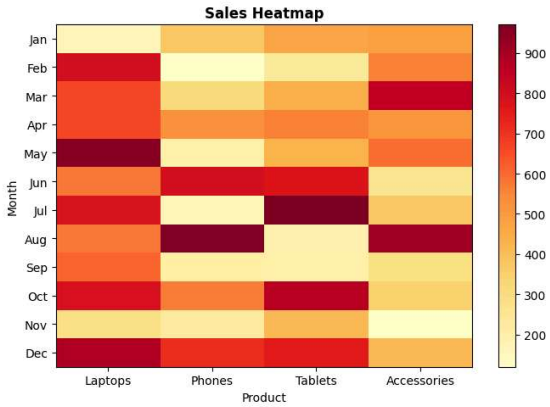
🎯 Practice Exercises

In [20]:
```
# 🎯 EXERCISE 1: Student Grade Analysis
print("🎯 EXERCISE 1: STUDENT GRADE ANALYSIS")
print("="*50)

# Create student data
np.random.seed(42)
students = pd.DataFrame({
    'Student_ID': range(1, 21),
    'Name': [f'Student_{i}' for i in range(1, 21)],
    'Math': np.random.randint(50, 100, 20),
    'Science': np.random.randint(60, 95, 20),
    'English': np.random.randint(55, 90, 20),
    'History': np.random.randint(65, 100, 20)
})

# Add calculated columns
students['Total'] = students[['Math', 'Science', 'English', 'History']].sum(axis
students['Average'] = students['Total'] / 4
students['Grade'] = pd.cut(students['Average'],
                           bins=[0, 60, 70, 80, 90, 100],
                           labels=['F', 'D', 'C', 'B', 'A'])

print("Student Dataframe:")
print(students.head())

# Analyze performance
print("\n\nGrade Distribution:")
print(students['Grade'].value_counts().sort_index())

print("\n\nSubject Averages:")
for subject in ['Math', 'Science', 'English', 'History']:
    print(f"{subject}: {students[subject].mean():.2f}")

# Find top 3 students
print("\n\nTop 3 Students:")
print(students.nlargest(3, 'Average')[['Name', 'Average', 'Grade']])
```

```
🎯 EXERCISE 1: STUDENT GRADE ANALYSIS
==================================================
Student Dataframe:
   Student_ID       Name  Math  Science  English  History  Total  Average  \
0           1  Student_1    88       89       80       79    336    84.00
1           2  Student_2    78       61       56       72    267    66.75
2           3  Student_3    64       80       74       78    296    74.00
3           4  Student_4    92       92       82       87    353    88.25
4           5  Student_5    57       71       61       85    274    68.50

  Grade
0     B
1     D
2     C
3     B
4     D


Grade Distribution:
Grade
F     0
D     5
C    10
B     5
A     0
Name: count, dtype: int64


Subject Averages:
Math: 72.85
Science: 76.05
English: 70.75
History: 80.65


Top 3 Students:
          Name  Average Grade
3    Student_4    88.25     B
6    Student_7    85.75     B
13  Student_14    85.25     B
```

In [21]:

```
# 🎯 EXERCISE 2: Sales Data Visualization
print("\n🎯 EXERCISE 2: SALES DATA VISUALIZATION")
print("="*50)

# Create sales data
months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
products = ['Laptops', 'Phones', 'Tablets', 'Accessories']

np.random.seed(42)
sales_data = {
    'Month': months * len(products),
    'Product': [product for product in products for _ in range(len(months))],
    'Sales': np.random.randint(100, 1000, len(months) * len(products)),
    'Revenue': np.random.randint(10000, 500000, len(months) * len(products))
}

sales_df = pd.DataFrame(sales_data)
print("Sales Data (first 10 rows):")
print(sales_df.head(10))
```

```
# Pivot for visualization
pivot_sales = sales_df.pivot_table(index='Month', columns='Product', values='Sal

# Create visualization
fig, axes = plt.subplots(2, 2, figsize=(14, 10))

# Plot 1: Line plot of monthly sales
for product in products:
    product_data = sales_df[sales_df['Product'] == product]
    axes[0, 0].plot(product_data['Month'], product_data['Sales'], marker='o', la
axes[0, 0].set_title('Monthly Sales by Product', fontweight='bold')
axes[0, 0].set_xlabel('Month')
axes[0, 0].set_ylabel('Sales')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)
plt.setp(axes[0, 0].xaxis.get_majorticklabels(), rotation=45)

# Plot 2: Bar chart of total sales per product
total_sales = sales_df.groupby('Product')['Sales'].sum()
bars = axes[0, 1].bar(total_sales.index, total_sales.values, color=['blue', 'gre
axes[0, 1].set_title('Total Sales per Product', fontweight='bold')
axes[0, 1].set_xlabel('Product')
axes[0, 1].set_ylabel('Total Sales')
for bar in bars:
    height = bar.get_height()
    axes[0, 1].text(bar.get_x() + bar.get_width()/2., height,
                    f'{int(height):,}', ha='center', va='bottom')

# Plot 3: Pie chart of revenue distribution
revenue_by_product = sales_df.groupby('Product')['Revenue'].sum()
axes[1, 0].pie(revenue_by_product.values, labels=revenue_by_product.index,
               autopct='%1.1f%%', startangle=90, colors=['#ff9999','#66b3ff','#9
axes[1, 0].set_title('Revenue Distribution by Product', fontweight='bold')

# Plot 4: Heatmap of sales data
im = axes[1, 1].imshow(pivot_sales.values, cmap='YlOrRd', aspect='auto')
axes[1, 1].set_title('Sales Heatmap', fontweight='bold')
axes[1, 1].set_xlabel('Product')
axes[1, 1].set_ylabel('Month')
axes[1, 1].set_xticks(range(len(products)))
axes[1, 1].set_xticklabels(products)
axes[1, 1].set_yticks(range(len(months)))
axes[1, 1].set_yticklabels(months)
plt.colorbar(im, ax=axes[1, 1])

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

```
🎯 EXERCISE 2: SALES DATA VISUALIZATION
==================================================
Sales Data (first 10 rows):
  Month  Product  Sales  Revenue
0   Jan  Laptops    202    77435
1   Feb  Laptops    535   329030
2   Mar  Laptops    960   338947
3   Apr  Laptops    370    41551
4   May  Laptops    206   388360
5   Jun  Laptops    171   283538
6   Jul  Laptops    800   341236
7   Aug  Laptops    120    13890
8   Sep  Laptops    714    51606
9   Oct  Laptops    221   499492
```