📊 STATISTICS FOR DATA SCIENCE

Section 3: Descriptive Statistics - Understanding Your Data

📚 Our Dataset: Student Exam Scores

Dataset Context: Final exam scores for 50 students in a Data Science class.

```
import numpy as np

# Simple dataset: 20 student scores (sample from 50)
scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

print(f"Student Scores: {scores}")
print(f"Total students: {len(scores)}")
print(f"Min score: {min(scores)}")
print(f"Max score: {max(scores)}")
```

Why this dataset: Simple, relatable, and shows all statistical concepts clearly.

3.1 Measures of Central Tendency

📍 Mean (Average)

Definition: Sum of all values divided by number of values.
Formula: Mean = Σx ÷ n

Example: *Scores: 65, 72, 68, 74, 85*
Sum = 65 + 72 + 68 + 74 + 85 = 364
Number of values = 5
Mean = 364 ÷ 5 = 72.8

📊 Mean Visualization

```
Scores: 65, 72, 68, 74, 85
        | | | | |
        ▼ ▼ ▼ ▼ ▼
┌────┬────┬────┬────┬────┐
│ 65 │ 72 │ 68 │ 74 │ 85 │
└────┴────┴────┴────┴────┘
     Add all → 364
   Divide by 5 → 72.8
           ↓
     MEAN = 72.8
```

Visual: All scores combine and average to a center point.

```
import numpy as np

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Simple mean calculation
mean_score = np.mean(scores)
print(f"Mean score: {mean_score}")
print(f"Calculation: Sum({sum(scores)}) / {len(scores)} =
{mean_score}")
```

🎯 Median

Definition: Middle value when data is sorted.

🤔 Why Both Mean and Median?

Mean: Affected by outliers. One billionaire increases average income.

Median: Not affected by outliers. Shows typical value.

Example: Salaries: [30k, 35k, 40k, 45k, 1M]

Mean = 230k (misleading), Median = 40k (accurate)

Example: *Scores: 45, 55, 58, 60, 62, 65, 68, 70, 72, 73, 75, 76, 77, 78, 80, 82, 85, 88, 90, 95*
Even number (20 scores) → Average of 10th & 11th values
10th value = 73, 11th value = 75
Median = (73 + 75) ÷ 2 = 74

📊 Median Visualization

```
Sorted Scores (20 students):
45 55 58 60 62 65 68 70 72 73 | 75 76 77 78 80 82 85 88 90
95
↑
MIDDLE
(between 73 and 75)

Median = (73 + 75) ÷ 2 = 74

50% below | 50% above
10 students | 10 students
```

Visual: Line divided equally in middle.

```
import numpy as np

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Simple median calculation
median_score = np.median(scores)
print(f"Median score: {median_score}")

# Show sorted scores
sorted_scores = sorted(scores)
print(f"\nSorted scores: {sorted_scores}")
print(f"Middle positions: {sorted_scores[9]} and
{sorted_scores[10]}")
print(f"Median = ({sorted_scores[9]} + {sorted_scores[10]}) / 2
= {median_score}")
```

🏆 Mode

Definition: Most frequent value in dataset.

Example: *Scores: 75, 76, 77, 75, 78, 75, 80*
75 appears 3 times (most frequent)
Mode = 75

📊 Mode Visualization

```
    Score : Frequency
75 : █████████ (3 times)
   76 : ███ (1 time)
   77 : ███ (1 time)
   78 : ███ (1 time)
   80 : ███ (1 time)

 Tallest bar → MODE = 75
```

Visual: Highest frequency bar.

```
from collections import Counter

scores = [75, 76, 77, 75, 78, 75, 80, 82, 75, 76]

# Simple mode calculation
score_counts = Counter(scores)
print("Score frequencies:")
for score, count in score_counts.items():
    print(f"  Score {score}: {count} times")

# Find mode
```

```
mode_score = max(score_counts, key=score_counts.get)
mode_count = score_counts[mode_score]

print(f"\nMode: {mode_score} (appears {mode_count} times)")
print(f"This is the most common score")
```

3.2 Measures of Dispersion (Spread)

📏 Range

Definition: Difference between highest and lowest values.
Formula: Range = Max - Min

Example: *Scores: 45, 55, 58, 60, 95*
Max = 95, Min = 45
Range = 95 - 45 = 50

📊 Range Visualization

```
0────────25────────50────────75────────100
                    |  |  |  |
                    ▼  |  |  ▼
                   45  |  |  95
 ├──────────────────────────────────────────┤
               RANGE = 50

             Min=45  Max=95
```

Visual: Distance from leftmost to rightmost point.

```
scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
```

```
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Simple range calculation
score_range = max(scores) - min(scores)
print(f"Minimum score: {min(scores)}")
print(f"Maximum score: {max(scores)}")
print(f"Range: {max(scores)} - {min(scores)} = {score_range}")
print(f"\nInterpretation: Scores span {score_range} points")
```

📊 Variance

Definition: Average of squared differences from mean.

Example: *Scores: 70, 80, 90*
Mean = 80
Differences: -10, 0, +10
Squared: 100, 0, 100
Average: (100+0+100)÷3 = 66.67
Variance = 66.67

📊 Variance Visualization

```
Mean = 80

Score: 70 80 90
Diff: -10 0 +10
Square: 100 0 100

Average squares = (100+0+100)/3 = 66.67

VARIANCE = 66.67
```

Visual: Squares show spread from center.

```
import numpy as np

scores = [70, 80, 90]

# Simple variance calculation
mean_score = np.mean(scores)
print(f"Mean: {mean_score}")
```

```
# Calculate squared differences
squared_diffs = [(x - mean_score) ** 2 for x in scores]
print(f"\nSquared differences from mean:")
for i, (score, sq_diff) in enumerate(zip(scores,
squared_diffs)):
    diff = score - mean_score
    print(f"  Score {score}: ({score} - {mean_score})² =
({diff})² = {sq_diff}")

# Calculate variance
variance = sum(squared_diffs) / len(scores)
print(f"\nVariance = Sum of squares / n")
print(f"         = {sum(squared_diffs)} / {len(scores)}")
print(f"         = {variance}")
```

📈 Standard Deviation

Definition: Square root of variance.
Formula: SD = √Variance

Example: *Variance = 225*
Standard Deviation = √225 = 15
Interpretation: Scores typically vary by 15 points from mean

📊 Standard Deviation Visualization

```
Mean = 75
SD = 10

55———65———75———85———95
| | | | |
-2SD -1SD Mean +1SD +2SD

68% within 1 SD (65 to 85)
95% within 2 SD (55 to 95)
```

Visual: Bell curve with standard deviation bands.

```
import numpy as np
import math

scores = [70, 80, 90]

# Calculate standard deviation
mean_score = np.mean(scores)
variance = np.var(scores)
std_dev = np.std(scores)
```

```
print(f"Mean: {mean_score}")
print(f"Variance: {variance}")
print(f"Standard Deviation: √{variance} = {std_dev}")
print(f"\nInterpretation:")
print(f"Scores typically vary by {std_dev:.1f} points from the
mean")
print(f"\nNormal distribution rule:")
print(f"68% within {mean_score-std_dev:.1f} to
{mean_score+std_dev:.1f}")
print(f"95% within {mean_score-2*std_dev:.1f} to
{mean_score+2*std_dev:.1f}")
```

📦 Interquartile Range (IQR)

Definition: Range of middle 50% of data.
Formula: IQR = Q3 - Q1

Example: *20 scores sorted*
Q1 (25th percentile) = 65
Q3 (75th percentile) = 82
IQR = 82 - 65 = 17

📊 IQR Visualization

```
Scores sorted:
45 55 58 60 62 | 65 68 70 72 73 75 76 77 78 80 | 82 85 88
90 95
Lowest 25% Middle 50% (IQR) Highest 25%
Q1=65 Q3=82
|----------------------|
IQR = 17
```

Visual: Middle section of sorted data.

```
import numpy as np

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Calculate IQR
q1 = np.percentile(scores, 25)
q3 = np.percentile(scores, 75)
iqr = q3 - q1
```

```
print(f"Q1 (25th percentile): {q1}")
print(f"Q3 (75th percentile): {q3}")
print(f"IQR: {q3} - {q1} = {iqr}")
print(f"\nInterpretation:")
print(f"Middle 50% of students scored between {q1} and {q3}")
print(f"This range of {iqr} points contains typical scores")

# Find outliers
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
outliers = [s for s in scores if s < lower_bound or s >
upper_bound]

print(f"\nOutlier bounds: {lower_bound:.1f} to
{upper_bound:.1f}")
print(f"Outliers: {outliers}")
```

3.3 Measures of Shape

🔄 Skewness

Definition: Measures asymmetry of distribution.

Type	Skewness	Mean vs Median	Example
Symmetric	≈ 0	Mean = Median	Normal test scores
Right-skewed	> 0	Mean > Median	Income (few rich people)
Left-skewed	< 0	Mean < Median	Exam (most score high)

📊 Skewness Visualization

```
Left Skew Symmetric Right Skew
█ █ █
███ ███ ███
█████ █████ █████
███████ ███████ ███████
█████████ █████████ █████████
← Tail Balanced Tail →
Mean < Median Mean = Median Mean > Median
```

```
from scipy.stats import skew
import numpy as np
```

```
# Different distributions
symmetric = [65, 70, 72, 75, 78, 80, 82]
right_skewed = [50, 60, 65, 70, 75, 85, 95]
left_skewed = [85, 88, 90, 92, 95, 98, 100]

print("Skewness Examples:")
print(f"Symmetric data: {skew(symmetric):.2f}")
print(f"Right-skewed: {skew(right_skewed):.2f}")
print(f"Left-skewed: {skew(left_skewed):.2f}")

print(f"\nOur student scores:")
scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]
print(f"Skewness: {skew(scores):.2f}")
print(f"Mean: {np.mean(scores):.1f}, Median:
{np.median(scores)}")
print("Slightly right-skewed (positive value)")
```

📉 Kurtosis

Definition: Measures "peakedness" and tail heaviness.

Type	Kurtosis	Description	Example
Mesokurtic	≈ 3	Normal peaks & tails	Standard tests
Leptokurtic	> 3	Sharp peak, heavy tails	Stock returns
Platykurtic	< 3	Flat peak, light tails	Uniform distribution

📊 Kurtosis Visualization

```
Platykurtic Mesokurtic Leptokurtic
███ █ █
█████ ███ ███
███████ █████ █████
█████████ ███████ ███████
Flat peak Normal peak Sharp peak
Light tails Normal tails Heavy tails
Kurtosis < 3 Kurtosis ≈ 3 Kurtosis > 3
```

```
from scipy.stats import kurtosis
import numpy as np

# Different distributions
normal_data = np.random.normal(0, 1, 1000)  # Normal
distribution
uniform_data = np.random.uniform(-3, 3, 1000)  # Uniform
distribution
```

```
print("Kurtosis Examples:")
print(f"Normal distribution: {kurtosis(normal_data,
fisher=False):.2f}")
print(f"Uniform distribution: {kurtosis(uniform_data,
fisher=False):.2f}")

print(f"\nOur student scores:")
scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]
kurt = kurtosis(scores, fisher=False)
print(f"Kurtosis: {kurt:.2f}")

if kurt > 3.5:
    print("Leptokurtic: Sharp peak, heavy tails")
elif kurt < 2.5:
    print("Platykurtic: Flat peak, light tails")
else:
    print("Mesokurtic: Normal distribution")
```

3.4 Percentiles and Quartiles

Percentile: Value below which a percentage of data falls.
Quartiles: Special percentiles dividing data into 4 equal parts.

📊 Percentiles Visualization

```
0%────25%────50%────75%────100%
| | | | |
Min Q1 Median Q3 Max
45 65 74 82 95

Quartiles divide data into 4 equal groups:
Group 1: 45-65 (lowest 25%)
Group 2: 65-74 (next 25%)
Group 3: 74-82 (next 25%)
Group 4: 82-95 (highest 25%)
```

```
import numpy as np

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Calculate percentiles
percentiles = [10, 25, 50, 75, 90, 95]
values = np.percentile(scores, percentiles)
```

```
print("=== PERCENTILE ANALYSIS ===")
for p, v in zip(percentiles, values):
    print(f"{p}th percentile: {v:.1f}")

print(f"\n=== QUARTILES ===")
print(f"Q1 (25th): {np.percentile(scores, 25):.1f}")
print(f"Q2 (50th, Median): {np.percentile(scores, 50):.1f}")
print(f"Q3 (75th): {np.percentile(scores, 75):.1f}")

print(f"\n=== INTERPRETATION ===")
print(f"If you score {np.percentile(scores, 90):.0f}, you're in top 10%")
print(f"If you score {np.percentile(scores, 75):.0f}, you're in top 25%")
print(f"Middle 50% scored between {np.percentile(scores, 25):.0f} and {np.percentile(scores, 75):.0f}")
```

3.5 Frequency Distribution

Definition: Shows how often each value or range occurs.

📊 Frequency Table

Score Range	Frequency	Percentage
40-49	1	5%
50-59	2	10%
60-69	4	20%
70-79	8	40%
80-89	3	15%
90-100	2	10%
Total	**20**	**100%**

Visual: Table showing counts in each range.

```
import pandas as pd

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]
```

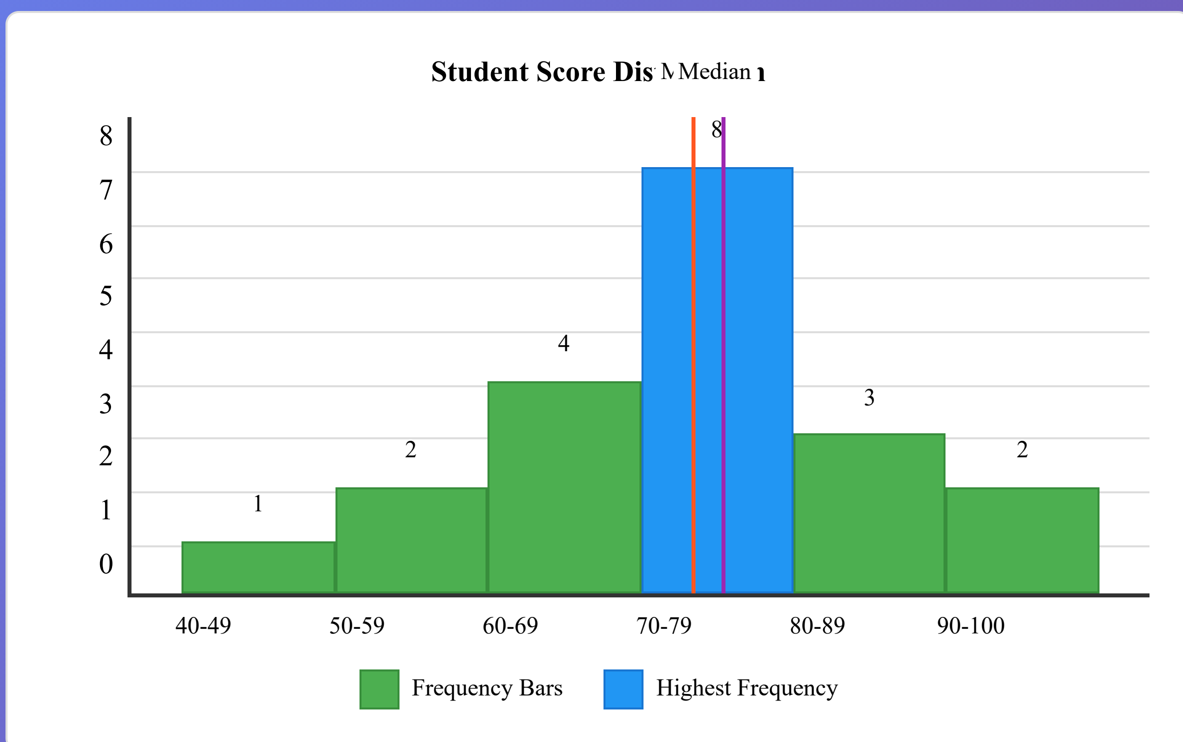
```
# Create frequency distribution
df = pd.DataFrame({'Score': scores})

# Create bins
bins = [40, 50, 60, 70, 80, 90, 101]
labels = ['40-49', '50-59', '60-69', '70-79', '80-89', '90-
100']

df['Range'] = pd.cut(df['Score'], bins=bins, labels=labels,
right=False)
freq_dist = df['Range'].value_counts().sort_index()

print("=== FREQUENCY DISTRIBUTION ===")
print("Range   | Frequency | Percentage")
print("-" * 35)

total = len(scores)
for r in labels:
    freq = freq_dist.get(r, 0)
    pct = (freq / total) * 100
    print(f"{r:7} | {freq:9} | {pct:8.1f}%")

print("-" * 35)
print(f"{'Total':7} | {total:9} | {100:8.1f}%")

print(f"\n=== INSIGHTS ===")
most_common = freq_dist.idxmax()
most_count = freq_dist.max()
print(f"Most common range: {most_common} ({most_count}
students)")
print(f"This contains {most_count/total*100:.1f}% of students")
```

3.6 Data Visualization Concepts

📊 Histogram

Definition: Bar graph showing frequency distribution of continuous data.

📊 ENHANCED HISTOGRAM VISUALIZATION



Interpretation: Shows distribution shape, central tendency, and spread at a glance.

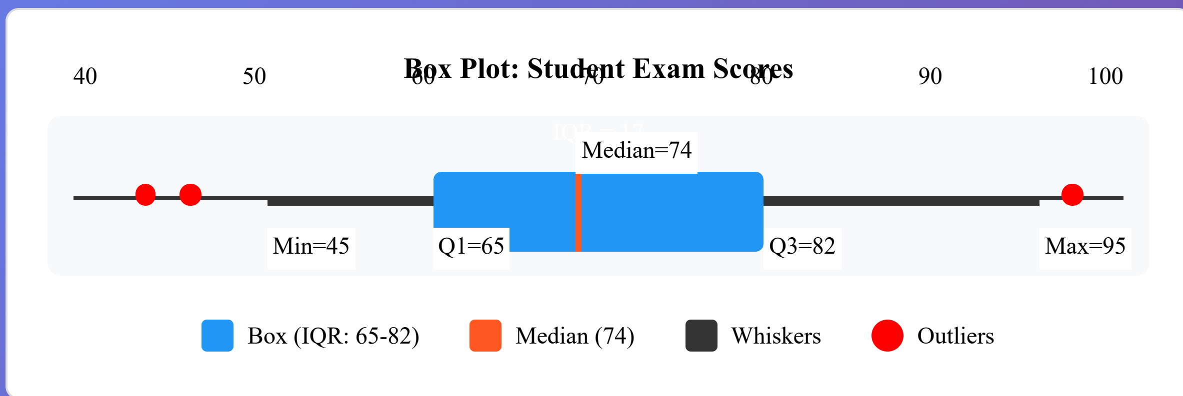
```
import matplotlib.pyplot as plt
import numpy as np
```

```
scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Create enhanced histogram
plt.figure(figsize=(10, 6))
n, bins, patches = plt.hist(scores, bins=6, edgecolor='black',
                            color='skyblue', alpha=0.7)

# Color the highest bar differently
max_freq_index = np.argmax(n)
patches[max_freq_index].set_facecolor('#2196F3')
patches[max_freq_index].set_edgecolor('#1976D2')
patches[max_freq_index].set_linewidth(2)

# Add mean and median lines
mean_score = np.mean(scores)
median_score = np.median(scores)
plt.axvline(mean_score, color='#FF5722', linestyle='--',
            linewidth=2, label=f'Mean = {mean_score:.1f}')
plt.axvline(median_score, color='#9C27B0', linestyle=':',
            linewidth=2, label=f'Median = {median_score}')

plt.xlabel('Exam Scores', fontsize=12)
plt.ylabel('Number of Students', fontsize=12)
plt.title('Enhanced Histogram: Score Distribution with Mean &
Median',
          fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3)

# Add value labels on bars
for i, (patch, count) in enumerate(zip(patches, n)):
    plt.text(patch.get_x() + patch.get_width()/2, count + 0.1,
             f'{int(count)}', ha='center', va='bottom',
fontsize=10)

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

📦 Box Plot (Box-and-Whisker Plot)

Definition: Visualizes distribution using quartiles, median, and outliers.

📦 ENHANCED BOX PLOT VISUALIZATION



Interpretation: Shows quartiles, median, range, and outliers in one view.

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

scores = [45, 55, 58, 60, 62, 65, 68, 70, 72, 73,
          75, 76, 77, 78, 80, 82, 85, 88, 90, 95]

# Create enhanced box plot
plt.figure(figsize=(10, 8))

# Create box plot with custom styling
box = plt.boxplot(scores, vert=True, patch_artist=True,
                  boxprops=dict(facecolor='#2196F3', alpha=0.8,
linewidth=2),
                  medianprops=dict(color='#FF5722',
linewidth=3),
                  whiskerprops=dict(color='#333', linewidth=2),
                  capprops=dict(color='#333', linewidth=2),
```

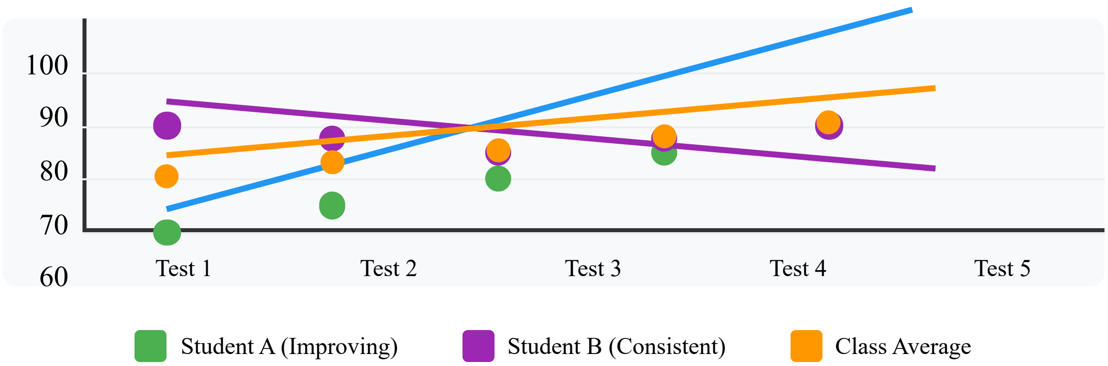
```
                  flierprops=dict(marker='o', color='#FF0000',
                                  markersize=10, alpha=0.7))

# Add individual data points
for i, score in enumerate(scores):
    plt.plot(1 + np.random.normal(0, 0.05), score, 'o',
             color='gray', alpha=0.5, markersize=6)

# Add annotations
q1 = np.percentile(scores, 25)
median = np.median(scores)
q3 = np.percentile(scores, 75)
iqr = q3 - q1

plt.text(1.15, q1, f'Q1 = {q1}', fontsize=11,
         bbox=dict(facecolor='white', alpha=0.9))
plt.text(1.15, median, f'Median = {median}', fontsize=11,
         bbox=dict(facecolor='white', alpha=0.9))
plt.text(1.15, q3, f'Q3 = {q3}', fontsize=11,
         bbox=dict(facecolor='white', alpha=0.9))
plt.text(1.3, (q1+q3)/2, f'IQR = {iqr}', fontsize=12,
fontweight='bold',
         bbox=dict(facecolor='#E3F2FD', alpha=0.9))

plt.ylabel('Exam Scores', fontsize=12)
plt.title('Enhanced Box Plot with Individual Data Points',
          fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3, axis='y')

# Remove x-ticks
plt.xticks([])

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

📈 Line Chart

Definition: Shows trends over time with connected points.

📈 ENHANCED LINE CHART VISUALIZATION

Student Performance Over 5 Tests



Trend Analysis: Student A shows steady improvement, while Student B remains consistent around 80-85

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

# Test scores over time
tests = ['Test 1', 'Test 2', 'Test 3', 'Test 4', 'Test 5']
student_a = [65, 70, 75, 78, 82]    # Improving student
student_b = [82, 80, 78, 79, 81]    # Consistent student
class_avg = [72, 74, 76, 78, 80]    # Class average

# Create enhanced line chart
plt.figure(figsize=(12, 7))
```

```
# Plot lines with different styles
line_a, = plt.plot(tests, student_a, 'o-', color='#4CAF50',
                   linewidth=3, markersize=10, label='Student A
(Improving)')
line_b, = plt.plot(tests, student_b, 's-', color='#9C27B0',
                   linewidth=2, markersize=8, label='Student B
(Consistent)')
line_avg, = plt.plot(tests, class_avg, '^-', color='#FF9800',
                     linewidth=4, markersize=12, label='Class
Average')

# Add value labels
for i, (test, score_a, score_b, score_avg) in
enumerate(zip(tests, student_a, student_b, class_avg)):
    plt.text(i, score_a + 1, f'{score_a}', ha='center',
va='bottom',
             fontsize=9, fontweight='bold', color='#4CAF50')
    plt.text(i, score_b - 2, f'{score_b}', ha='center',
va='top',
             fontsize=9, fontweight='bold', color='#9C27B0')
    plt.text(i, score_avg + 1.5, f'{score_avg}', ha='center',
va='bottom',
             fontsize=10, fontweight='bold', color='#FF9800')

# Add trend lines and annotations
plt.annotate('Steady Improvement', xy=(2, 75), xytext=(1, 85),
             arrowprops=dict(arrowstyle='->', color='#4CAF50',
linewidth=2),
             fontsize=11, fontweight='bold', color='#4CAF50',
             bbox=dict(facecolor='white', alpha=0.9))

plt.annotate('Consistent Performance', xy=(2, 78), xytext=(3,
70),
             arrowprops=dict(arrowstyle='->', color='#9C27B0',
linewidth=2),
             fontsize=11, fontweight='bold', color='#9C27B0',
             bbox=dict(facecolor='white', alpha=0.9))

plt.xlabel('Test Number', fontsize=12)
plt.ylabel('Score', fontsize=12)
```

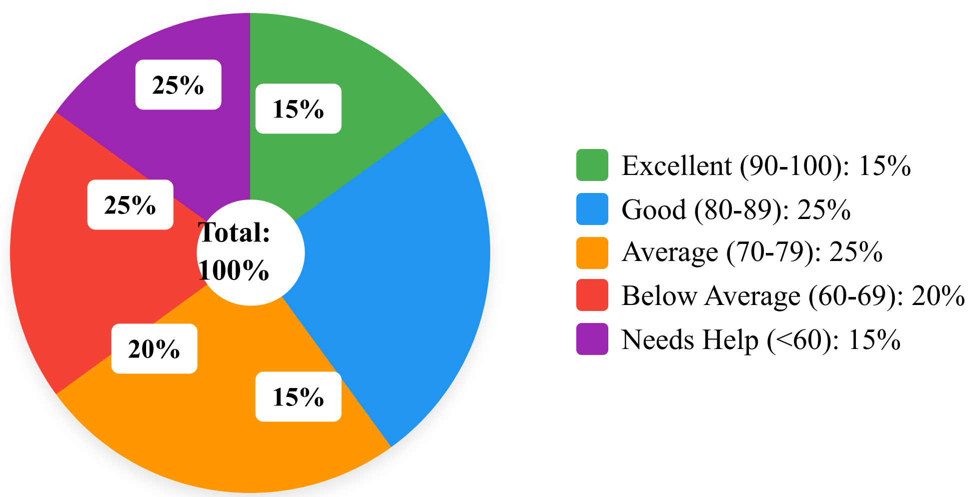
```python
plt.title('Enhanced Line Chart: Student Performance Trends',
          fontsize=14, fontweight='bold')
plt.legend(loc='lower right', fontsize=11)
plt.grid(True, alpha=0.3, linestyle='--')
plt.ylim(60, 90)

# Add overall trend
plt.text(2.5, 62, 'Overall: Gradual improvement across tests',
         ha='center', fontsize=11, fontstyle='italic',
         bbox=dict(facecolor='#E3F2FD', alpha=0.9))

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

🥧 Pie Chart

Definition: Circle divided into proportional slices showing parts of a whole.

🥧 ENHANCED PIE CHART VISUALIZATION

Student Performance Distribution



Key Insights:

- 50% of students scored 70 or above (Good + Excellent)
- 25% of students need additional support
- Performance follows roughly normal distribution

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

# Performance categories
categories = ['Excellent (90-100)', 'Good (80-89)', 'Average
```

```
(70-79)',
              'Below Average (60-69)', 'Needs Help (<60)']
students = [3, 5, 5, 4, 3]  # 20 students total
colors = ['#4CAF50', '#2196F3', '#FF9800', '#F44336',
'#9C27B0']
explode = (0.05, 0.05, 0.05, 0.05, 0.05)  # Slight separation

# Create enhanced pie chart
plt.figure(figsize=(12, 10))

# Create pie chart
wedges, texts, autotexts = plt.pie(students, labels=categories,
colors=colors,
                                   autopct='%1.1f%%',
startangle=90,
                                   explode=explode,
shadow=True,
                                   textprops={'fontsize': 11})

# Enhance wedge properties
for wedge in wedges:
    wedge.set_edgecolor('white')
    wedge.set_linewidth(2)

# Style percentage texts
for autotext in autotexts:
    autotext.set_color('white')
    autotext.set_fontsize(10)
    autotext.set_fontweight('bold')

# Style category labels
for text in texts:
    text.set_fontsize(11)
    text.set_fontweight('bold')

# Add title
plt.title('Enhanced Pie Chart: Student Performance
Distribution\n(20 Students Total)',
          fontsize=14, fontweight='bold', pad=20)
```

```
# Add center circle
centre_circle = plt.Circle((0,0), 0.4, color='white',
linewidth=2)
plt.gca().add_artist(centre_circle)

# Add center text
plt.text(0, 0, 'Total\n100%', ha='center', va='center',
         fontsize=12, fontweight='bold', color='#333')

# Add legend with counts
legend_labels = [f'{cat}: {count} students' for cat, count in
zip(categories, students)]
plt.legend(wedges, legend_labels, title="Performance Details",
           loc="center left", bbox_to_anchor=(1, 0.5),
           fontsize=10)

# Add insights box
insight_text = "Key Insights:\n• 50% scored 70+\n• 25% need
support\n• Normal distribution"
plt.text(1.8, -0.8, insight_text, fontsize=10,
         bbox=dict(facecolor='#E3F2FD', alpha=0.9,
boxstyle='round,pad=0.5'))

plt.axis('equal')  # Equal aspect ratio ensures pie is drawn as
circle
plt.tight_layout()
plt.show()
```

📊 Bar Chart

Definition: Rectangular bars showing categorical data values.

📊 ENHANCED BAR CHART VISUALIZATION

Student Grades Distribution by Subject

8	6	9	4	7
Math	**Science**	**English**	**History**	**Computer**

10
8
6
4
2
0

Analysis:

- **Highest:** English (9 students with Grade A)
- **Lowest:** History (4 students with Grade A)
- **Average:** 6.8 students per subject with Grade A
- English and Math are strongest subjects

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

# Subjects and number of students with Grade A
```

```
subjects = ['Math', 'Science', 'English', 'History', 'Computer
Science']
grade_a_counts = [8, 6, 9, 4, 7]
colors = ['#4CAF50', '#2196F3', '#FF9800', '#9C27B0',
'#F44336']

# Create enhanced bar chart
plt.figure(figsize=(12, 8))

# Create bars with gradient effect
bars = plt.bar(subjects, grade_a_counts, color=colors,
edgecolor='black',
               linewidth=2, alpha=0.9)

# Add gradient effect to bars
for bar, color in zip(bars, colors):
    # Create gradient effect
    gradient = np.linspace(0.8, 1.0, 100).reshape(1, -1)
    gradient = np.vstack((gradient, gradient))

    # Get bar position
    x = bar.get_x()
    width = bar.get_width()
    height = bar.get_height()

    # Create gradient rectangle
    plt.imshow(gradient, extent=[x, x+width, 0, height],
               aspect='auto', cmap=plt.cm.Blues_r if 'blue' in
color else
               plt.cm.Greens_r if 'green' in color else
               plt.cm.Oranges_r if 'orange' in color else
               plt.cm.Purples_r if 'purple' in color else
               plt.cm.Reds_r, alpha=0.6)

# Add value labels on top of bars
for i, (bar, count) in enumerate(zip(bars, grade_a_counts)):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height + 0.2,
             f'{count}', ha='center', va='bottom', fontsize=12,
fontweight='bold')
```

```
    # Add percentage label inside bar
    percentage = (count / sum(grade_a_counts)) * 100
    plt.text(bar.get_x() + bar.get_width()/2., height/2,
             f'{percentage:.1f}%', ha='center', va='center',
             color='white', fontsize=10, fontweight='bold')

# Add average line
average = np.mean(grade_a_counts)
plt.axhline(y=average, color='red', linestyle='--',
linewidth=2,
            label=f'Average = {average:.1f}')

plt.xlabel('Subjects', fontsize=12)
plt.ylabel('Number of Students with Grade A', fontsize=12)
plt.title('Enhanced Bar Chart: Grade A Distribution Across
Subjects',
          fontsize=14, fontweight='bold')
plt.legend()
plt.grid(True, alpha=0.3, axis='y')

# Add insights annotation
insights = f"""Analysis:
• Highest: English (9 students)
• Lowest: History (4 students)
• Average: {average:.1f} students/subject
• Total: {sum(grade_a_counts)} students with Grade A"""

plt.text(0.02, 0.98, insights, transform=plt.gca().transAxes,
         fontsize=10, verticalalignment='top',
         bbox=dict(boxstyle='round', facecolor='#FFF3E0',
alpha=0.9))

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

📋 Summary & Key Takeaways

Measure	When to Use	Example	Python Function
Mean	No outliers, normal data	Average temperature	np.mean()
Median	With outliers, skewed data	House prices, salaries	np.median()
Mode	Categorical data, most common	Favorite color, size	statistics.mode()
Range	Quick spread measure	Temperature variation	max()-min()
Variance	Statistical calculations	Quality control	np.var()
Standard Deviation	Interpret spread in units	Test score spread	np.std()
IQR	With outliers, typical range	Salary typical range	np.percentile()

🎯 Quick Decision Guide:

1. **Start with visualization - Histogram or box plot first**
2. **Check for outliers - Use IQR method**
3. **Choose central tendency: Mean for symmetric, Median for skewed**
4. **Report with dispersion: "Mean = 70, SD = 10" not just "Mean = 70"**

5. **Visualize to communicate: A good chart beats 1000 numbers**

📊 The Complete Picture

```
DESCRIPTIVE STATISTICS
┌─────────────────────────────────────┐
│ 1. Central Tendency │
│ • Mean (center) │
│ • Median (middle) │
│ • Mode (most frequent) │
│ │
│ 2. Dispersion (Spread) │
│ • Range (total spread) │
│ • Variance (average squares) │
│ • Standard Deviation (√variance) │
│ • IQR (middle 50%) │
│ │
│ 3. Shape │
│ • Skewness (asymmetry) │
│ • Kurtosis (peakedness) │
│ │
│ 4. Visualization │
│ • Histogram (distribution) │
│ • Box plot (quartiles) │
│ • Bar chart (categories) │
│ • Pie chart (proportions) │
│ • Line chart (trends) │
└─────────────────────────────────────┘
```

Data Science with Vamsi

Next: Section 4 - Inferential Statistics