University of Sulaimani
College of Science
Computer Department
4th Stage

Data Science Management

Statistics for Data Science

Class 4

Theoretical and practical lectures

سؤال مهم في الامتحان :- لماذا نستخدم (linear - Algebra) في Data Science ؟

Assist. Prof. Dr. Miran Taha Abdullah
2025-2026

Class Agenda

- Introduction to Statistics in Data Science
- Descriptive statistics
- Inferential statistics
- Python Demonstrations

Objective

- Gain a clear understanding of descriptive and inferential statistics and their relevance to data science and computer science.
- Be able to apply key statistical methods in real-world data science problems using Python.
- Learn to use both descriptive and inferential statistics to extract insights, validate hypotheses, and make data-driven decisions.

Introduction to Statistics in Data Science

Why Statistics Matter in Data Science:

- In data science, **statistics** are essential for **analyzing** and **interpreting** data. From **cleaning** data to building **predictive** models, statistical techniques form the backbone of data-driven decisions.

  Examples:

  - **AI Models** (training data, performance evaluation),
  - **Machine Learning** (feature selection, model validation),
  - **Big Data** (trend analysis, anomaly detection).

- **Descriptive statistics** help summarize data for immediate insights.

مثل average و max و min

(conclusion) مثل خاتمة بحث تقرير

- **Inferential statistics** enable us to make predictions and test hypotheses beyond the sample.

فرضيات

- Python libraries like **Pandas**, **SciPy**, **Statsmodels**, and **Scikit-learn** are essential for **performing statistical analysis** in data science.

- Understanding both branches of statistics is crucial for building **accurate**, **reliable models** in computer science and machine learning.

Pandas: Used for **data manipulation and exploration**. It provides powerful data structures like DataFrames to clean, organize, and summarize data before analysis.

Example: Calculating mean, median, or grouping data by categories.

SciPY: Provides **advanced scientific and statistical functions**. It's useful for probability distributions, hypothesis testing, and numerical integration.

Example: Performing t-tests, chi-square tests, or computing correlation coefficients.

Statsmodels – Specialized for **statistical modeling and inference**. It supports regression analysis, time series analysis, and statistical tests.

Example: Building linear regression models or performing ANOVA tests.

Scikit-learn: Primarily used for **machine learning**, but also supports statistical tasks like regression, classification, and clustering. It's great for predictive modeling using statistical techniques.

Example: Predicting outcomes with linear regression or detecting anomalies using clustering algorithms.

Key Concepts: Descriptive Statistics

في الامتحان يجيبلك سؤال وهو مسئلة ويقلك اكتب تحليل؟ منك لازم تستخدم

1- Measures of Central Tendency:

- **Mean**: The average of all data points.
- **Median**: The middle value when data is sorted.
- **Mode**: The most frequently occurring value.

2- Measures of Dispersion:

- **Range**: Difference between the maximum and minimum values.
- **Variance**: Average of the squared differences from the mean.
- **Standard Deviation**: Square root of the variance; measures data spread.

3- Skewness and Kurtosis:

- **Skewness**: Describes the asymmetry of the data distribution.
- **Kurtosis**: Measures the "tailedness" of the data distribution.

Key Concepts: Inferential Statistics

1- Sampling and Sampling Distribution:

- Importance of sample size and randomness.
- Central Limit Theorem

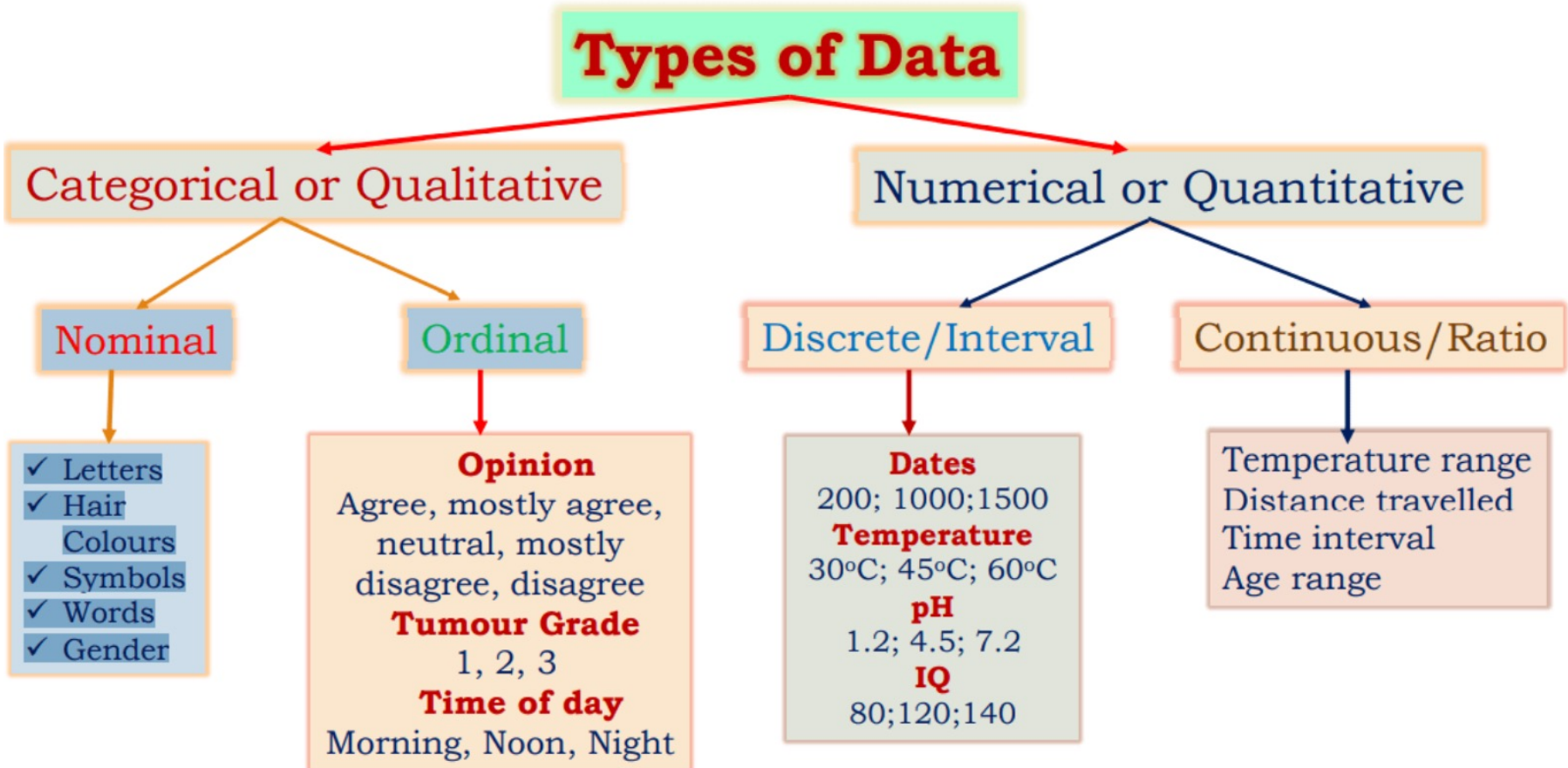
2- Hypothesis Testing:

- **Null Hypothesis (H_0)**
- **Alternative Hypothesis (H_1)**
- **p-value**
- **Types of Tests: t-tests** (one-sample, independent, paired). **ANOVA**: Comparing means across multiple groups. **Chi-square Test**: Testing categorical variables.

3- Confidence Intervals:

4- Regression Analysis:

- **Linear Regression**: Predicting continuous outcomes.
- **Logistic Regression**: Predicting binary outcomes.



1- Measures of Central Tendency:

These help to describe the center of the data.

Mean: The average of all data points.

- *Example*: The average exam score of 50 students is 80 out of 100.

Median: The middle value when the data points are ordered.

- *Example*: For the exam scores, if the sorted values are [50, 60, 70, 80, 90], the median score is 70.

Mode: The most frequent value in the dataset.

- *Example*: In a set of [1, 2, 2, 3, 3, 3, 4], the mode is 3 because it occurs most frequently

The Mean

- The mean is a common and intuitive way to summarize a set of numbers it might simply called the “average”. The mean is the sum of all of the data elements divided by how many elements there are

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n} : \frac{\sum_{i=1}^{n} x_i}{n}$$

When you use the sum() function on a list of numbers, it adds up all the numbers in the list and returns the result.

```
>>> shortlist = [1, 2, 3]
>>> sum(shortlist)
6
```

We can use the len() function to give us the length of a list:

```
>>> len(shortlist)
3
```

Mean = 6 / 3 → 2.0

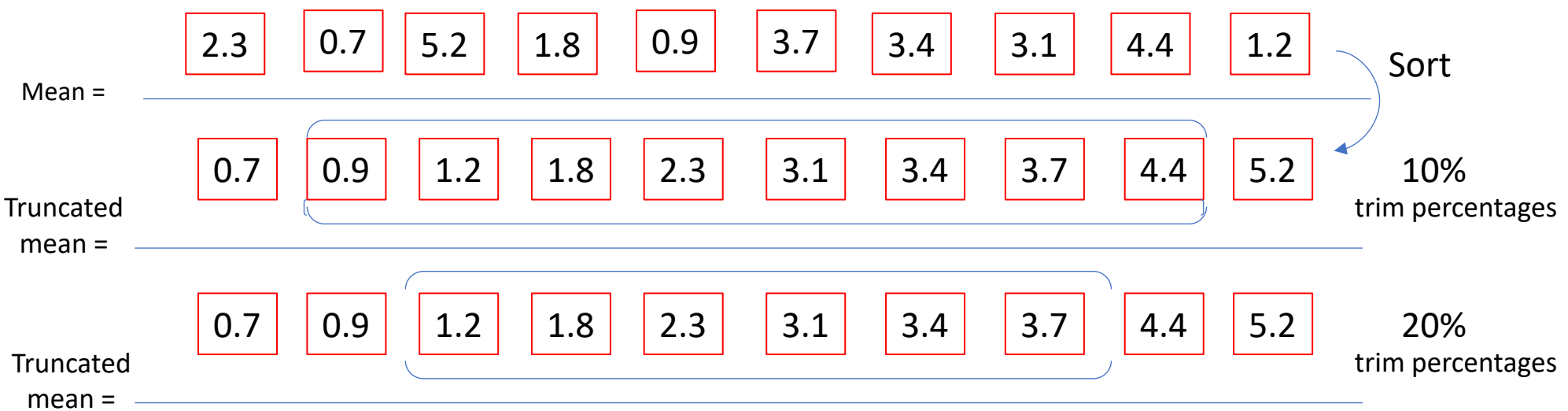
Example

Say there's a school charity that's been taking donations over a period of time spanning the last 12 days (we'll refer to this as period A). In that time, the following 12 numbers represent the total dollar amount of donations received for each day: 100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, and 1200. We can calculate the mean by summing these totals and then dividing the sum by the number of days. In this case, the sum of the numbers is 5733. If we divide this number by 12 (the number of days), we get 477.75, which is the mean donation per day. This number gives us a general idea of how much money was donated on any given day.

```
def calculate_mean(numbers):
    s = sum(numbers)
    N = len(numbers)
# Calculate the mean
    mean = s/N
     return mean
if __name__ == '__main__':
    donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]
    mean = calculate_mean(donations)
    N = len(donations)
    print('Mean donation over the last {0} days is {1}'.format(N, mean))
```

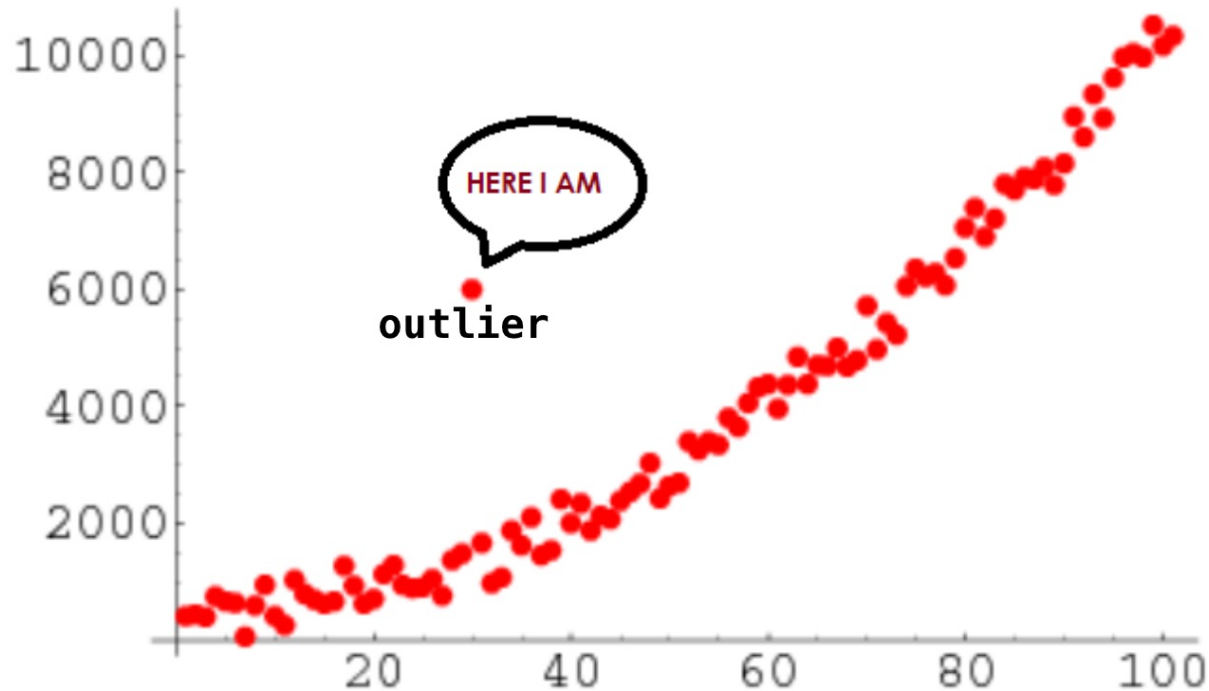
Mean donation over the last 12 days is 477.75

A trimmed mean (similar to an adjusted mean) is a method of averaging that removes a small designated percentage of the largest and smallest values before calculating the mean. After removing the specified **outlier** observations(switch to next slide) , the trimmed mean is found using a standard arithmetic averaging formula. The use of a trimmed mean helps eliminate the influence of outliers or data points on the tails that may unfairly affect the traditional or arithmetic mean.



$$\text{Trimmed mean} = \bar{x} = \frac{\sum_{i=p+1}^{n-p} x_{(i)}}{n - 2p}$$

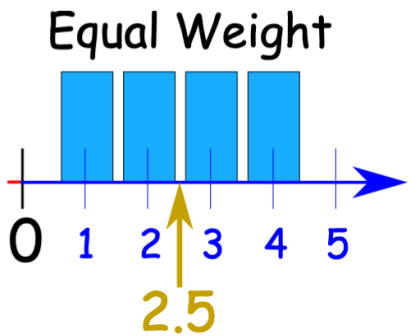
An **outlier** is a data point in a data set that is distant from all other observations. A data point that lies outside the overall distribution of the dataset



- What causes the outliers?
- Impact of the outlier
- Methods to Identify outliers

The weighted mean is a type of mean that is calculated by multiplying the weight (or probability) associated with a particular event or outcome with its associated quantitative outcome and then summing all the products together.

$$\text{Weighted mean} = \bar{x}_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i}^{n} w_i}$$



$$\text{Mean} = \frac{1+2+3+4}{4} = \frac{10}{4} = 2.5$$

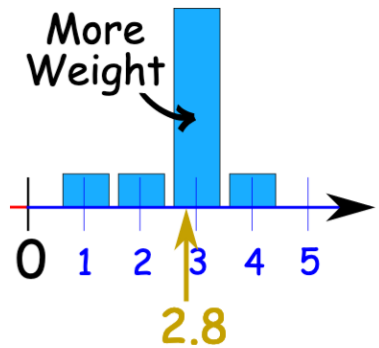
Weights

Each of those numbers has a "weight" of ¼"

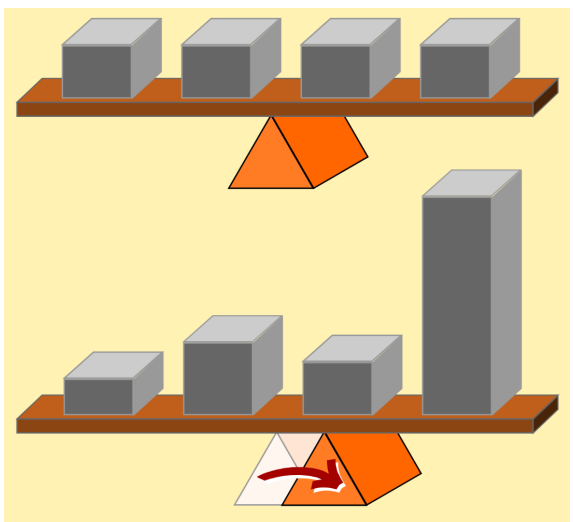
$$\text{Mean} = \tfrac{1}{4} \times 1 + \tfrac{1}{4} \times 2 + \tfrac{1}{4} \times 3 + \tfrac{1}{4} \times 4$$
$$= 0.25 + 0.5 + 0.75 + 1 = \mathbf{2.5}$$

Now let's change the weight of **3** to 0.7, and the weights of the other numbers to 0.1

$$\text{Mean} = 0.1 \times 1 + 0.1 \times 2 + 0.7 \times 3 + 0.1 \times 4$$
$$= 0.1 + 0.2 + 2.1 + 0.4 = \mathbf{2.8}$$



When some values get more weight than others,
the central point (the mean) can change



Example: Sam wants to buy a new camera, and decides on the following rating system:

- Image Quality **50%**
- Battery Life **30%**
- Zoom Range **20%**

The Sonu camera gets 8 (out of 10) for Image Quality, 6 for Battery Life and 7 for Zoom Range

The **Conan camera** gets 9 for Image Quality, 4 for Battery Life and 6 for Zoom Range

Which camera is best?

Sonu: $0.5 \times 8 + 0.3 \times 6 + 0.2 \times 7 = 4 + 1.8 + 1.4 =$ **7.2**
Conan: $0.5 \times 9 + 0.3 \times 4 + 0.2 \times 6 = 4.5 + 1.2 + 1.2 =$ **6.9**

Sam decides to buy the Sonu.

Median

- The **median** of a collection of numbers is another kind of average. To find the median, we sort the numbers in ascending order. If the length of the list of numbers is odd, the number in the middle of the list is the median. If the length of the list of numbers is even, we get the median by taking the mean of the two middle numbers.

```
>>> samplelist = [4, 1, 3]
>>> samplelist.sort()
>>> samplelist
[1, 3, 4]
```

[100, 60,70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]

Sort

[60, 70,100, 100, 200, 500, 500, 503, 600, 900, 1000, 1200] ***Even***

(500+500)/2

[60, 70, 100, 100, 200, 500, 500, 503, 600, 800, 900, 1000, 1200] ***Odd***

500

```
import statistics

data = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200]

# Calculate median
median_value = statistics.median(data)
print(median_value)
```

1. Sort the data:

[60, 70, 100, 100, 200, 500, 500, 503, 600, 900, 1000, 1200]

2.Count the number of elements: 12 (even number)

2.Median for even number = **average of the middle two values**:
→ Middle values: 500 and 500
→ Median = (500 + 500) / 2 = **500**

```
def calculate_median(numbers):
    N = len(numbers)
    numbers.sort()
    # Find the median
     if N % 2 == 0:
         # if N is even
          m1 = N/2
          m2 = (N/2) + 1
          # Convert to integer, match position
           m1 = int(m1) - 1
           m2 = int(m2) - 1
           median = (numbers[m1] + numbers[m2])
        else:
            m = (N+1)/2
            # Convert to integer, match position
            m = int(m) - 1
             median = numbers[m]
          return median
if __name__ == '__main__':
    donations = [100, 60, 70, 900, 100, 200, 500, 500, 503, 600, 1000, 1200
    median = calculate_median(donations)
    N = len(donations)
    print('Median donation over the last {0} days is {1}'.format(N, median))
```

صنوال

Finding the Mode and Creating a Frequency Table

Instead of finding the mean value or the median value of a set of numbers, what if you wanted to find the number that occurs most frequently? This number is called the mode.

Consider the test scores of a math test (out of 10 points) in a class of 20 students: 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, and 10. The mode of this list would tell you which score was the most common in the class. There's no symbolic formula for calculating the mode—you simply count how many times each unique number occurs and find the one that occurs the most.

The most_common() method of the Counter class

```
# Import Counter from collections
from collections import Counter

# Sample list of numbers
simplelist = [4, 2, 1, 3, 4]

# Create a Counter object to count occurrences
c = Counter(simplelist)
```

```
# 1. Most common element (top 1)
print(c.most_common(1))
# Output: [(4, 2)]

# 2. All elements sorted by frequency
print(c.most_common())
# Output: [(4, 2), (2, 1), (1, 1), (3, 1)]

# 3. Top 2 most common elements
print(c.most_common(2))
# Output: [(4, 2), (2, 1)]
```

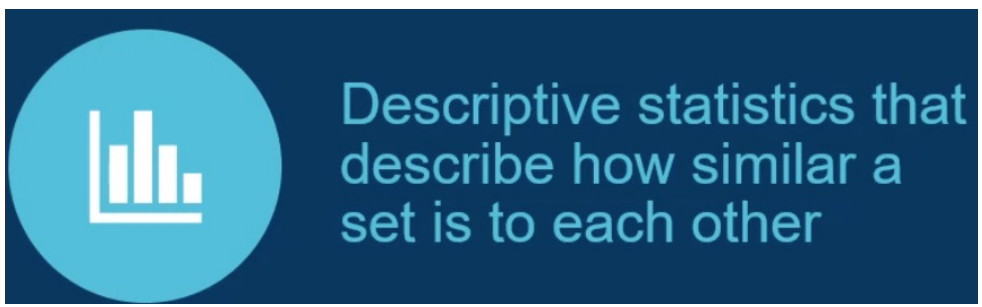
مكرر - مرام قال جيب > بس واحد هو المتكرر اذن الثاني راح يكون اي بعد المتكرر

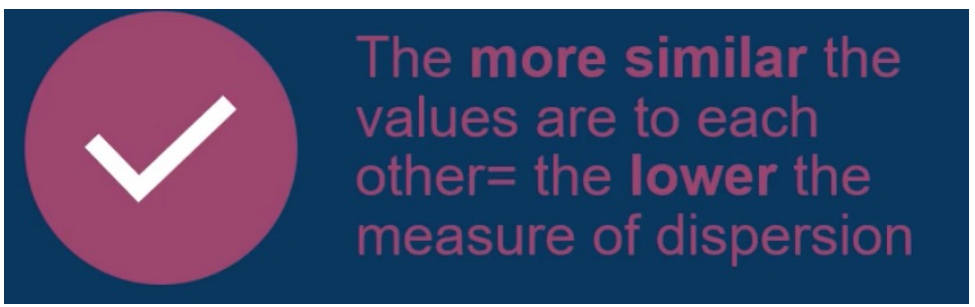
What if you have a set of data where two or more numbers occur the same maximum number of times? For example, in the list of numbers 5, 5, 5, 4, 4, 4, 9, 1, and 3, both 4 and 5 are present three times. In such cases, the list of numbers is said to have multiple modes, and the program should find and print all the modes.

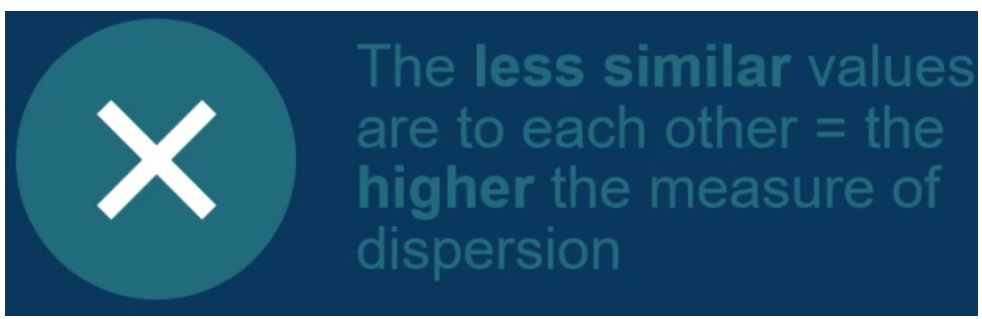
```
'''
Calculating the mode when the list of numbers may
have multiple modes
'''


from collections import Counter


def calculate_mode(numbers):

    c = Counter(numbers)
    numbers_freq = c.most_common()
    max_count = numbers_freq[0][1]

    modes = []
    for num in numbers_freq:
        if num[1] == max_count:
            modes.append(num[0])
    return modes

if __name__ == '__main__':
    scores = [5, 5, 5, 4, 4, 4, 9, 1, 3]
    modes = calculate_mode(scores)
    print('The mode(s) of the list of numbers are:')
    for mode in modes:
        print(mode)
```

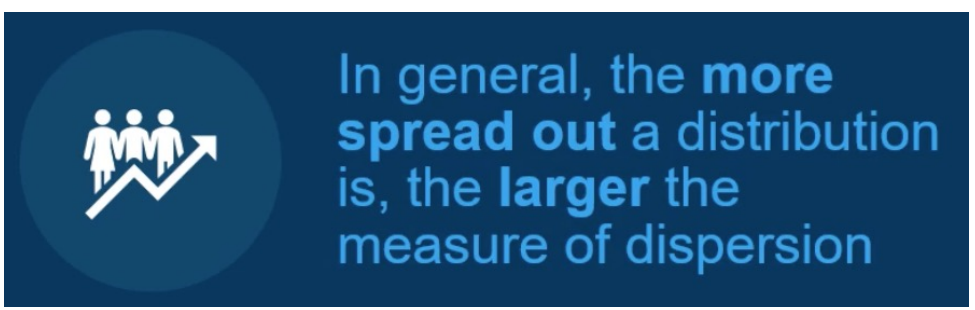
2- Measuring the Dispersion

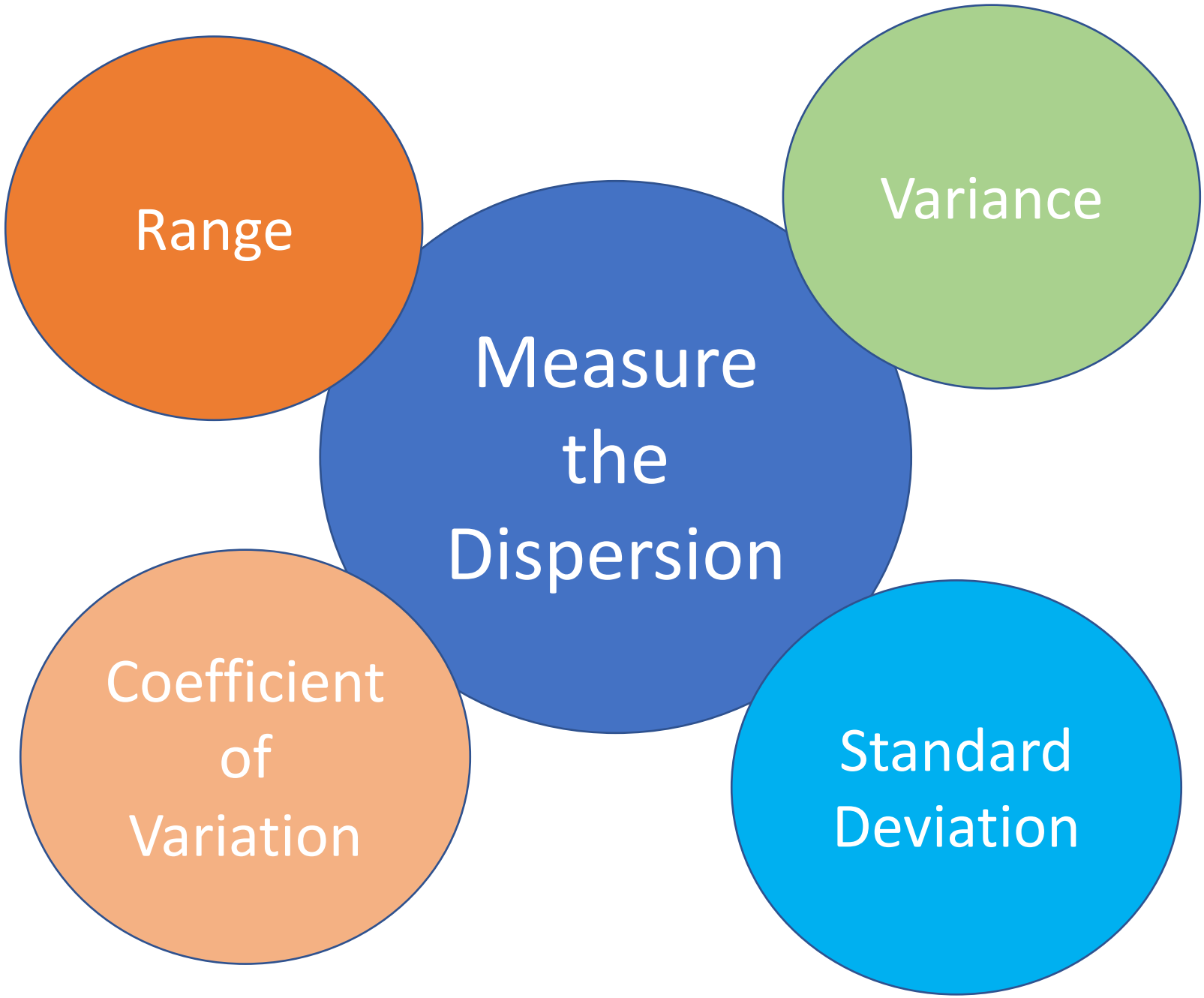
Measuring the dispersion tells us how far away the numbers in a set of data are from the mean of the data set. We'll learn to calculate three different measurements of dispersion: range, variance, and standard deviation.











Range: The difference between the maximum and minimum values.

- *Example*: If the exam scores range from 55 to 95, the range is 95 - 55 = 40.

Variance: The average of the squared differences from the mean.

Standard Deviation: The square root of the variance. It indicates how spread out the data points are.

- *Example*: If the exam scores are tightly clustered around the mean, the standard deviation will be low. If the scores are more spread out, the standard deviation will be high.

Coefficient of Variation (CV) is a standardized measure of dispersion that shows how much variability exists in relation to the mean of the dataset.

Finding the Range of a Set of Numbers

The **Range** is the **difference between the largest and smallest values** in a dataset.

$$\text{Range} = \text{Maximum Value} - \text{Minimum Value}$$

Benefit:
It gives a **quick idea of how spread out the data** is.

Example:
If students' test scores range from **40 to 90**, → **Range** = 90 − 40 = **50**

This means there is a **50-point spread** between the lowest and highest score.

Interpretation:
A **larger range** = more variation;
a **smaller range** = data values are more consistent.

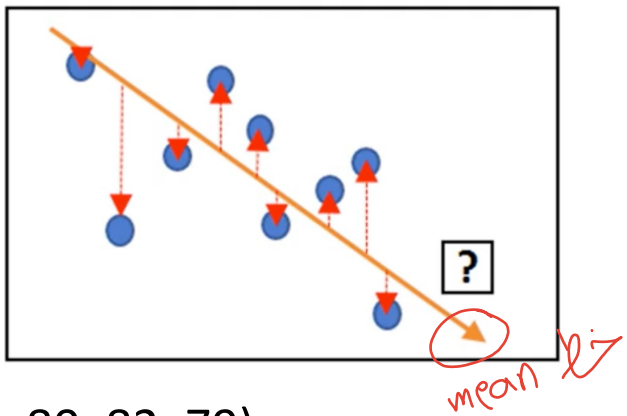
Finding the Variance

Variance measures **how far each data point is from the mean**, on average, by taking the **average of squared differences** from the mean.

$$\sigma^2 = \frac{\sum_{i=1}^{n} \left(x_i - \overline{x}\right)^2}{N}$$

x_i = Each value in the data set
$\overline{x}$ = Mean of all values in the data set
N = Number of values in the data set

Benefit:
Shows **data spread** and **consistency** of data points.



Example:
Suppose two teachers give weekly quizzes.
In Class A, students' scores are close to each other (e.g., 78, 80, 82, 79).
In Class B, scores vary widely (e.g., 50, 70, 90, 100).
Class A has a **smaller variance**, meaning students' performance is **more consistent.**

Interpretation:
High variance → data points vary widely (more fluctuation).
Low variance → data points are close to the mean (Consistent data).

Finding the Standard deviation (SD)

The standard deviation is calculated as the square root of variance by determining each data point's deviation relative to the mean.

Benefit:
Easier to interpret than variance because it's in the **same units** as the original data.

Example:
In exam scores, if SD is **low**, most students performed close to the average.

If SD is **high**, performance varied a lot.

Interpretation:
Small SD → consistent performance.
Large SD → large differences among values.

$$\text{Standard Deviation} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}}$$

where:

x_i = Value of the i^{th} point in the data set

$\overline{x}$ = The mean value of the data set

n = The number of data points in the data

Finding the Co-efficient of Variation (CV)

The coefficient of variation (relative standard deviation) is commonly used to compare the data dispersion between distinct series of data

$$\text{CV} = \frac{\sigma}{\mu}$$

where:

$\sigma = \text{standard deviation}$

$\mu = \text{mean}$

Benefit:

Used to **compare variability** across datasets with different scales or units.

Example:

Machine A: Mean output = 100 kg, SD = 5 → CV = 5%

Machine B: Mean output = 200 kg, SD = 40 → CV = 20%

Machine A is **more consistent**.

Interpretation:

A **smaller CV** means **more stability** and **less risk**.

Calculating the Correlation Between Two Data Sets

Correlation measures the **strength and direction of a relationship** between two variables.
It ranges from **–1 to +1**.

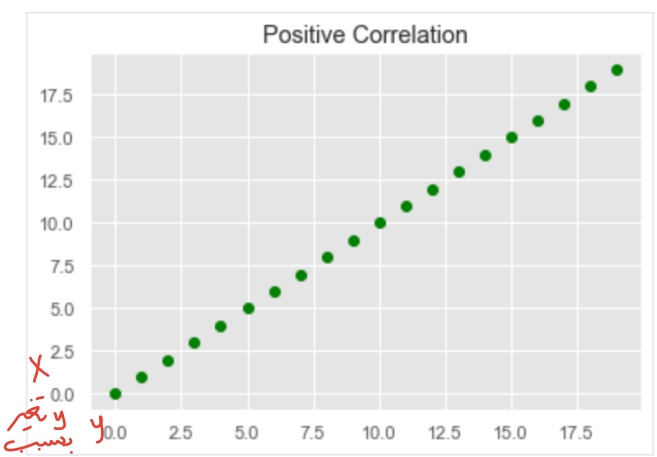
+1 → Perfect positive correlation
0 → No correlation
–1 → Perfect negative correlation

يجيب سؤال وانت لازم تزعم وتبين العلاقة بين x و y ؟

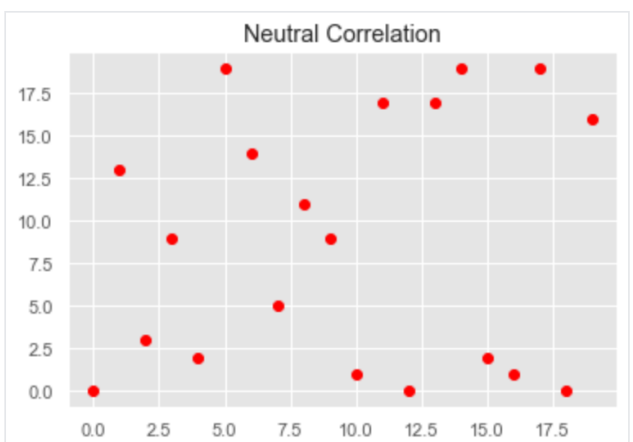
Benefit:
Helps in **predicting** one variable based on another.

Example:

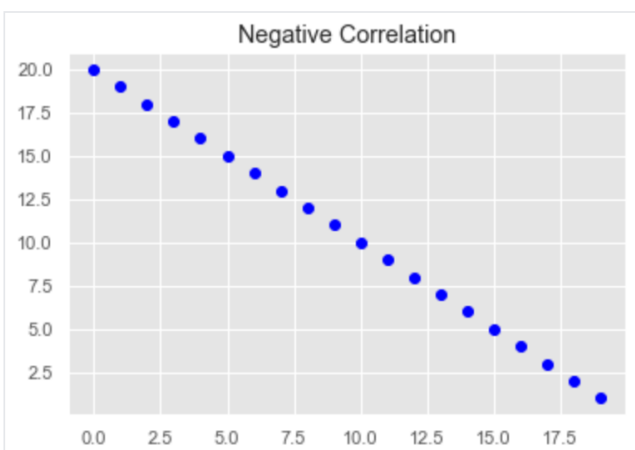
r = 0.9 → strong positive relation



r = 0 → no linear relationship



r = –0.8 → strong negative relation



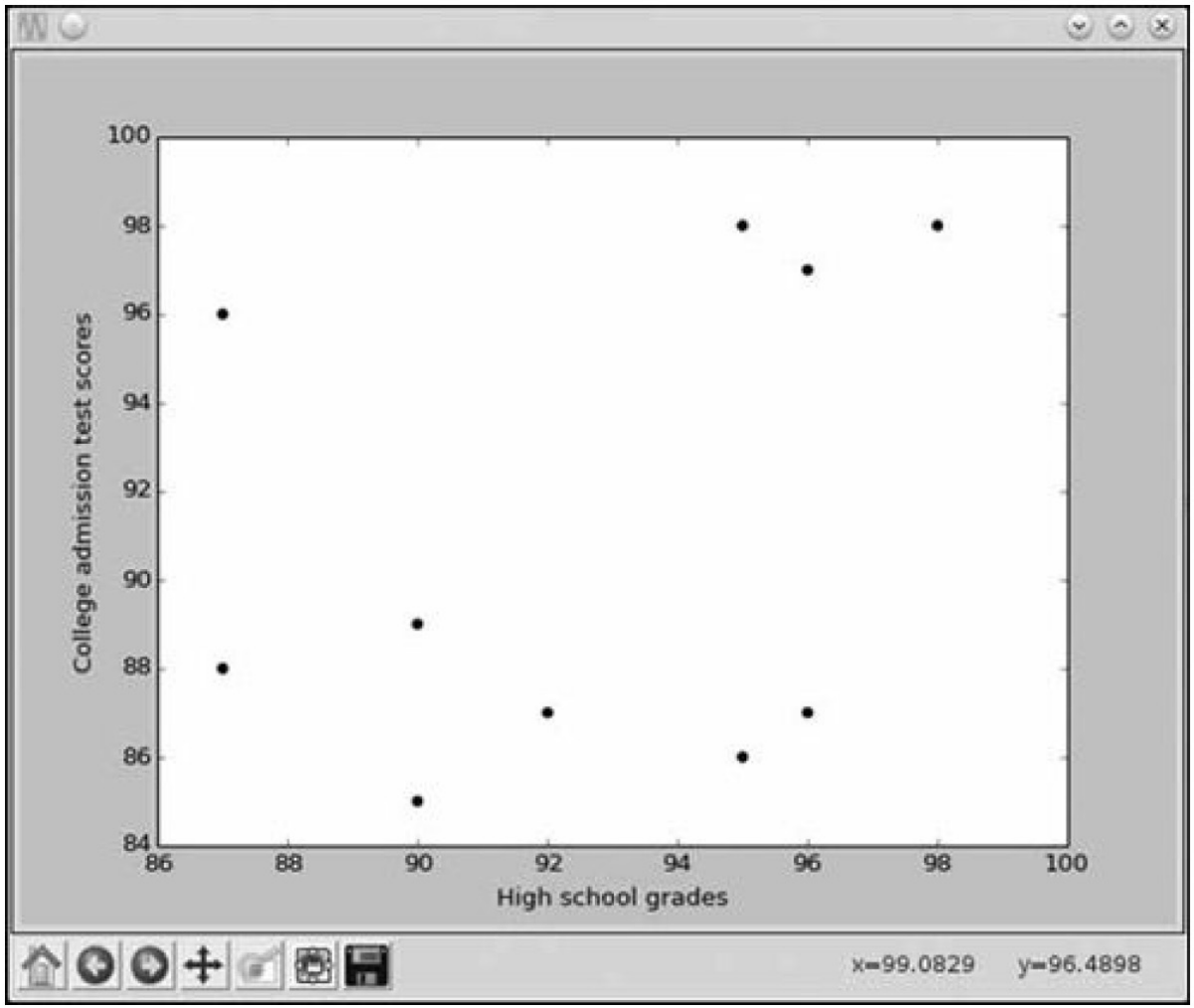
$$\text{correlation} = \frac{n\sum xy - \sum x \sum y}{\sqrt{\left(n\sum x^2 - (\sum x)^2\right)\left(n\sum y^2 - (\sum y)^2\right)}}.$$

Σxy Sum of the products of the individual elements of the two sets of numbers, x and y
Σx Sum of the numbers in set x
Σy Sum of the numbers in set y
(Σx)2 Square of the sum of the numbers in set x
(Σy)2 Square of the sum of the numbers in set y
Σx2 Sum of the squares of the numbers in set x
Σy2 Sum of the squares of the numbers in set y

High School Grades and Performance on College Admission Tests

High school grades	College admission test scores
90	85
92	87
95	86
96	97
87	96
87	88
90	89
95	98
98	98
96	87

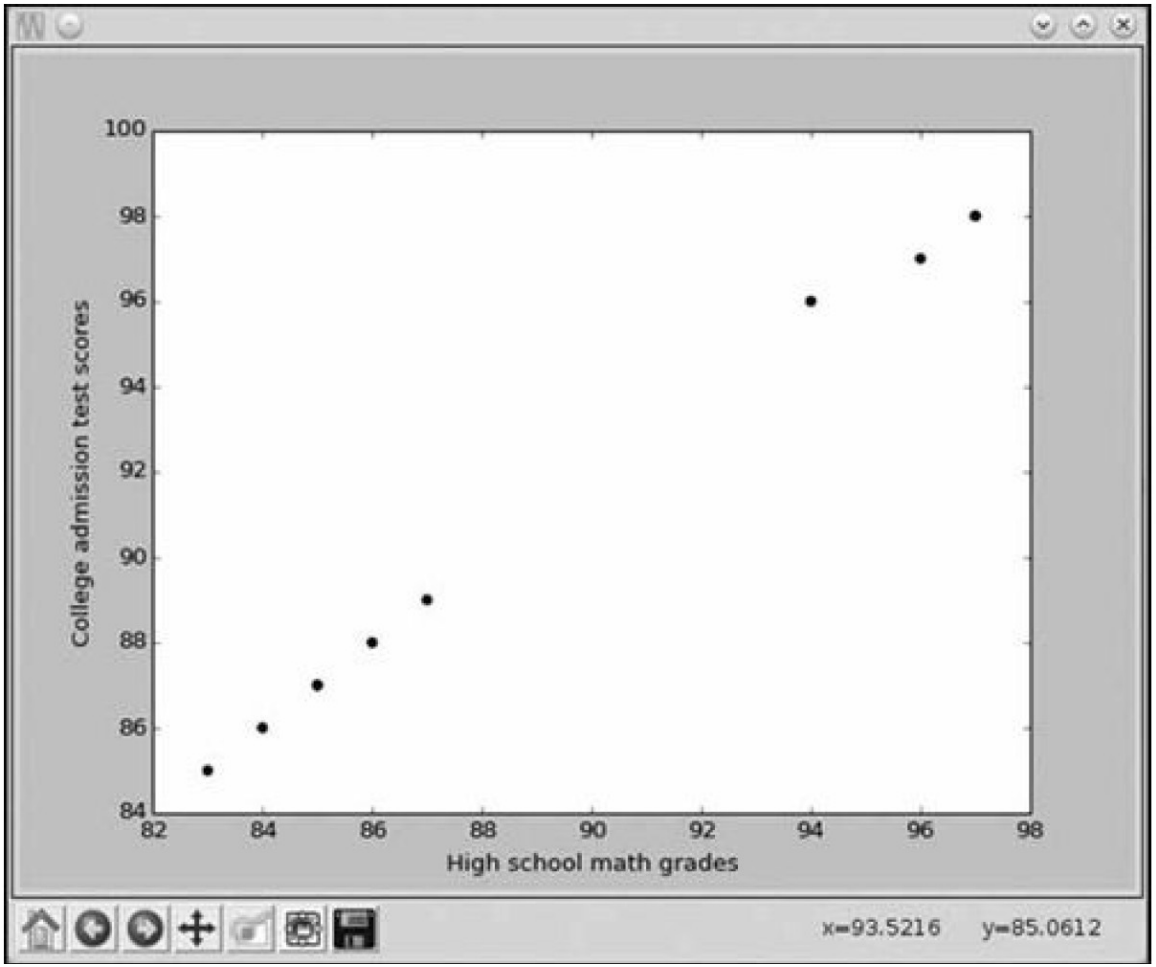
Find correlation between high school grades and college admission test scores



We see that it's approximately **0.32**. This means that there's **some correlation**, but **not a very strong one**. **If the correlation were closer to 1**, we'd see this reflected in the scatter plot as well—the points would conform more closely to a straight, diagonal line.

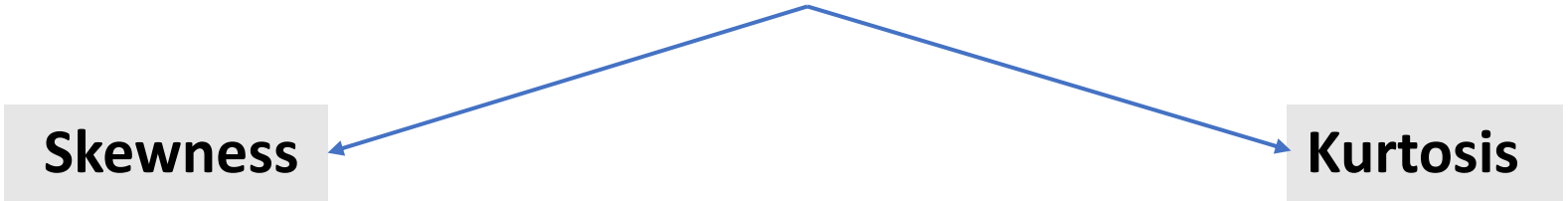
High School Math Grades and Performance on College Admission Tests

High school math grades	College admission test scores
83	85
85	87
84	86
96	97
94	96
86	88
87	89
97	98
97	98
85	87



Along a straight line. This is an indication of a high correlation between the high school math scores and performance on the college admission test. The correlation coefficient, in this case, turns out to be approximately 1. With the help of the scatter plot and correlation coefficient, we can conclude that there is indeed a **strong relationship** in this data set between grades in high school math and performance on college admission tests.

3- Skewness and Kurtosis:



Skewness tells us if a dataset is symmetric or if it leans to one side.

Importance: Helps understand if most values are low or high, and detect bias in data.

Example:

Dataset 1: [2, 3, 4, 5, 6] → symmetric → skewness ≈ 0

Dataset 2: [1, 2, 2, 3, 10] → right-skewed → skewness > 0

Interpretation*:* Most numbers are small, but one very high number (10) pulls the tail to the right.

Kurtosis shows how heavy or light the tails of a distribution are.

Importance: Helps detect extreme values (outliers) that can affect analysis.

Example:

Dataset 1: [3, 3, 4, 4, 5] → low kurtosis (flat, few extremes)
Dataset 2: [1, 3, 4, 5, 10] → high kurtosis (peaked, extreme values present)

Interpretation: Dataset 2 has more extreme values (1 and 10) than Dataset 1.

Python (scipy.stats.skew, scipy.stats.kurtosis) to calculate these values quickly.

Problem:

A teacher recorded the scores of 7 students in a quiz:

Scores: 5, 6, 7, 8, 8, 9, 15

We want to analyze the **shape of the distribution** using Skewness and Kurtosis.

Step 1: Calculate basic statistics:

Mean (average) = (5+6+7+8+8+9+15)/7 = 8
Median = middle value = 8
Standard deviation (SD) ≈ 3.13

Step 2: Skewness:
Skewness formula

$$\text{Skewness} = \frac{\text{Mean} - \text{Median}}{\text{SD}}$$

Skewness : (8 – 8)/3.13 ≈ 0 → approximately symmetric but notice the high value 15 pulls the tail slightly right.

Interpretation:
Skewness ≈ 0 → fairly symmetric distribution
High value 15 → small right skew

Step 3: Kurtosis

Kurtosis measures how heavy the tails are.

High kurtosis → more extreme values.
most scores are 5–9, but 15 is an extreme value → heavy right tail.

So kurtosis is positive → indicates the presence of outlier.

Step 4: Conclusion / Result

Skewness: Slightly positive → tail on the right
Kurtosis: Positive → one extreme high score (outlier)

Skewness tells which side the data is stretched.
Kurtosis tells whether there are extreme values (outliers)

Descriptive Statistics

- numpy.sum() - Computes the sum of array elements.
- numpy.mean() : Computes the arithmetic mean of the array elements.
- numpy.median() : Computes the median of the array elements.
- numpy.average() : Computes the weighted average of the array elements.
- numpy.std() : Computes the standard deviation of the array elements.
- numpy.var() : Computes the variance of the array elements.
- numpy.ptp() : Computes the range (max - min) of values in the array.
- numpy.min() : Finds the minimum value in the array.
- numpy.max() : Finds the maximum value in the array.
- numpy.percentile() : Computes the nth percentile of the array elements.
- numpy.quantile() : Computes the quantile of the array elements.

Frequency and Histogram

- numpy.histogram() : Computes the histogram of the data.
- numpy.bincount() : Counts the number of occurrences of each value in an array of non-negative integers.

Correlation and Covariance

- numpy.corrcoef() : Computes the Pearson correlation coefficient matrix.
- numpy.cov() : Computes the covariance matrix.

Statistical Testing

- numpy.random.normal() : Generates random samples from a normal distribution (useful for statistical simulations).
- numpy.random.seed() : Sets the random seed for reproducibility.

Random Number Generation

- numpy.random.seed() : Set random seed.
- numpy.random.rand() : Random numbers in [0, 1).
- numpy.random.randn() : Samples from a standard normal distribution.
- numpy.random.randint() : Random integers in a range.
- numpy.random.choice() : Random selection from an array.
- numpy.random.shuffle() : Shuffle elements of an array.
- numpy.random.normal() : Samples from a normal distribution.
- numpy.random.uniform() : Samples from a uniform distribution.

Exercise: Analyzing Historical Gold and Silver Prices

The closing prices of **Gold (USD/oz)** and **Silver (USD/oz)** over 7 years are:

Central Tendency
Calculate the **mean**, **median**, and **mode** for Gold and Silver prices over these years.

Dispersion
Calculate the **range**, **variance**, and **standard deviation** for both metals.

Skewness
Calculate the **skewness** and determine if the price distributions are symmetric, right-skewed, or left-skewed.

Kurtosis
Calculate the **kurtosis** and comment on the presence of extreme price movements (heavy tails).

Interpretation
Compare Gold and Silver in terms of **average price, volatility, and risk**.

Year	Gold Price (USD/oz)	Silver Price (USD/oz)
2017	1250	16.8
2018	1265	15.7
2019	1390	17.0
2020	1760	20.5
2021	1820	24.0
2022	1800	21.5
2023	1950	23.5

Introduction to Inferential Statistics

Inferential statistics uses a sample of data to make estimates, predictions, or decisions about a population.

Purpose: Go beyond descriptive statistics to draw conclusions about a larger group.

Importance:

- Helps make decisions under uncertainty.
- Enables prediction for unobserved data.
- Basis for hypothesis testing and confidence intervals.

Example:

A data scientist surveys 50 students' exam scores to estimate the average score of all 500 students in a university.

Key Parameters and Units

Parameter	Definition	Unit	Purpose/Reason
Population Mean (μ)	Average of all population values	Same as data (e.g., points, cm, $)	Represents true average of the population
Sample Mean ($\bar{x}$)	Average of sample values	Same as data	Used to estimate μ
Population Variance (σ^2)	Measure of population spread	Square of data unit (e.g., points^2)	True variability in population
Sample Variance (s^2)	Measure of sample spread	Square of data unit	Estimates σ^2 when population data unknown
Standard Deviation (σ / s)	Square root of variance	Same as data	Easier interpretation of spread
Confidence Interval (CI)	Range where population parameter likely lies	Same as data	Quantifies uncertainty in estimation
Significance Level (α)	Probability of Type I error	None (0–1)	Controls risk of rejecting true hypothesis
p-value	Probability of observing data if H_o is true	None (0–1)	Helps decide whether to reject H_o

Types of Inferential Statistics

Estimation

Point Estimation: Single value estimate (e.g., sample mean = 72 points).

Interval Estimation (Confidence Intervals): Range estimate (e.g., $\mu = 72 \pm 5$ points, 95% CI).

Reason to Use: Gives measure of certainty about parameter.

Hypothesis Testing

Null Hypothesis (H_0): Statement of no effect (e.g., mean score = 70).

Alternative Hypothesis (H_1): Statement of effect (e.g., mean score ≠ 70).

t-test / z-test: Compare sample mean with population mean.

Chi-square test: Test association between categorical variables.

Reason to Use: Decide if observed effect is statistically significant.

Regression and Correlation

Correlation: Measures strength of relationship (unitless, -1 to 1).

Simple Linear Regression: Predict Y from X ($Y = a + bX$).

Reason to Use: Model relationships and make predictions.

ANOVA (Analysis of Variance)

Compare means of multiple groups.

F-statistic: Ratio of between-group to within-group variance.

Reason to Use: Detect if group differences are significant.

1) Point Estimation

Definition: Point estimation refers to the process of using data from a sample to estimate a population parameter, such as the **mean** or **variance**.

```
import numpy as np

# Sample data: Heights of 10 students in cm
heights = [150, 160, 155, 165, 170, 175, 160, 158, 162, 168]

# Calculate sample mean
sample_mean = np.mean(heights)

print(f"Estimated population mean height: {sample_mean:.2f} cm")
```

Estimated population mean height: **161.30 cm**

2) Confidence Intervals

Definition: Confidence Interval is a range of values that is used to estimate a population parameter. We use the sample mean and standard deviation to calculate a range in which the true population parameter is likely to lie.

Confidence Level: The probability that the confidence interval contains the true parameter. Common confidence levels are **90%**, **95%**, and **99%**.

$$\text{CI} = \hat{\mu} \pm Z \times \frac{\sigma}{\sqrt{n}}$$

$\hat{\mu}$ = sample mean

Z = Z-score corresponding to the confidence level

σ = sample standard deviation

n = sample size

```
z_score = stats.norm.ppf(1 - (1 - confidence_level) / 2)
```

Scenario

Imagine you're conducting a survey to estimate the average height of 100 students in a classroom. However, you only measure the heights of a small sample of 10 students. Based on the **sample**, you want to **estimate** the **average height** of all the students in the classroom.

Number of Students : 10 Students

Range of height : 150 - 168 cm

```
import scipy.stats as stats

# Sample data
sample_data = [150, 160, 155, 165, 170, 175, 160, 158, 162, 168]

# Sample statistics
sample_mean = np.mean(sample_data)
sample_std = np.std(sample_data, ddof=1)
n = len(sample_data)

# Confidence level (95%)
confidence_level = 0.95
z_score = stats.norm.ppf(1 - (1 - confidence_level) / 2)                    Percent Point Function

# Calculate margin of error
margin_of_error = z_score * (sample_std / np.sqrt(n))

# Calculate confidence interval
ci_lower = sample_mean - margin_of_error
ci_upper = sample_mean + margin_of_error

print(f"Confidence Interval: ({ci_lower:.2f}, {ci_upper:.2f}) cm")
```

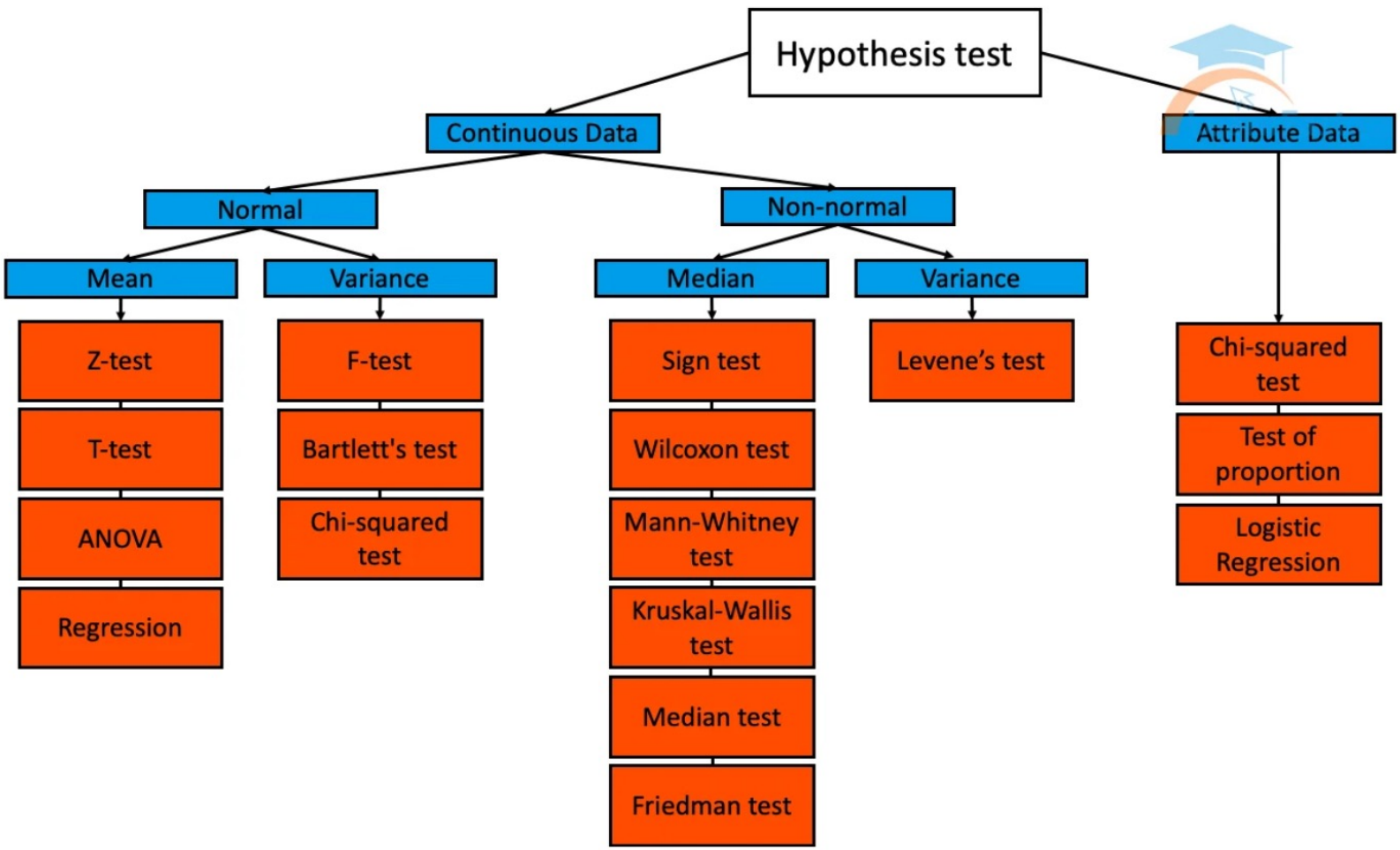
Confidence Interval: (157.61, 164.99) cm

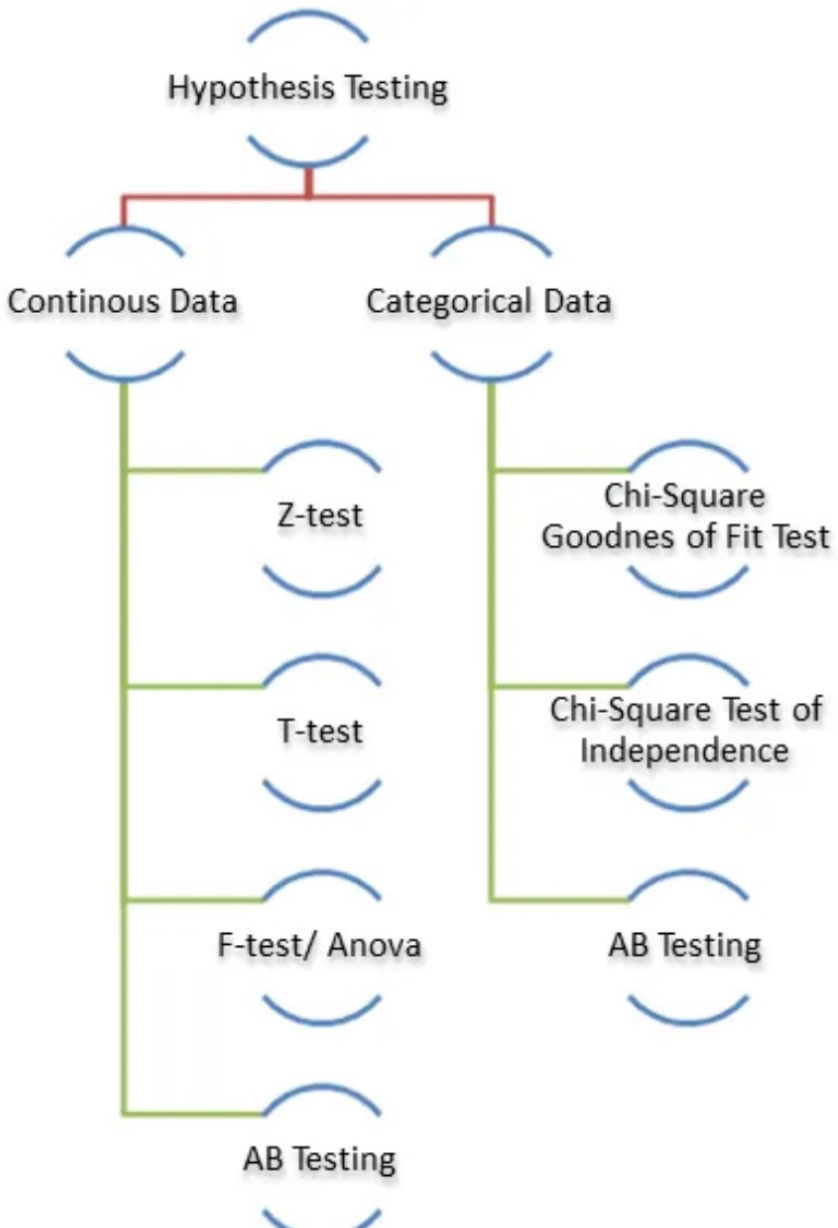
Scenario

You are a teacher at a school, and you want to estimate the average score of all students in your class on a final exam. You decide to randomly sample 40 students and calculate their average score. Based on the sample, you will construct a **95% Confidence Interval** for the true average exam score of all students in the class.

3) Hypothesis Testing

Definition: Hypothesis testing allows us to test an assumption or claim about a population based on sample data. We test the **null hypothesis** (H_0) against the **alternative hypothesis** (H_1).

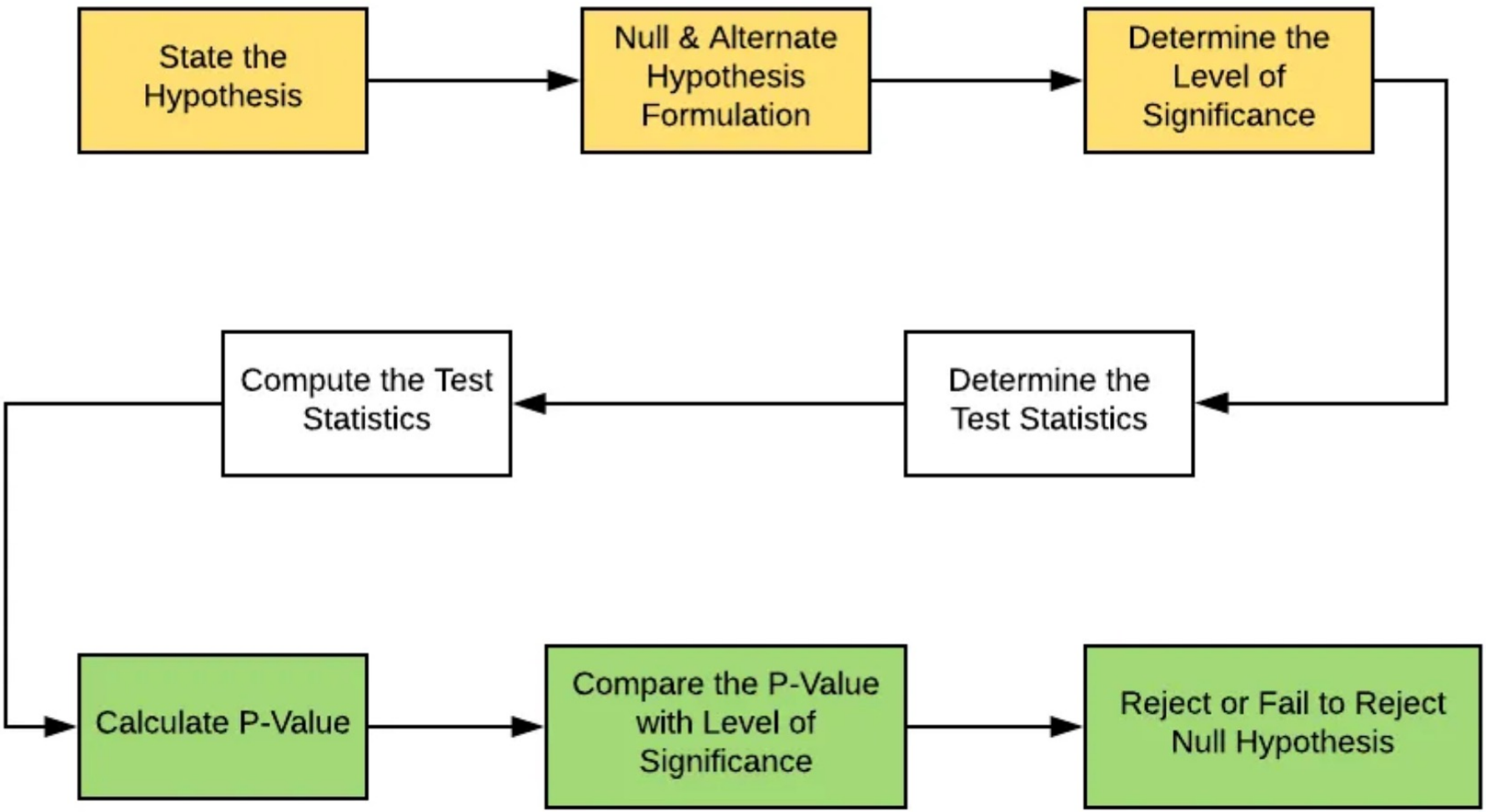




ID	Gender	Favorite Fruit	Education Level
1	Male	Apple	Bachelor
2	Female	Banana	Master
3	Female	Orange	PhD
4	Male	Apple	High School

ID	Age	Height (cm)	Weight (kg)
1	25	175	70
2	30	160	55
3	22	180	80
4	28	165	65

found out of it.



Hypothesis Testing

Definition: Hypothesis testing allows us to test an assumption or claim about a population based on sample data. We test the **null hypothesis** (H_0) against the **alternative hypothesis** (H_1).

Hypothesis Testing:

1. State the Hypotheses:

1. **Null Hypothesis (H_0)**: The population parameter is equal to a specific value.
2. **Alternative Hypothesis (H_1)**: The population parameter is not equal to that value (or different in a specific way).

2. Set the Significance Level (α):

1. The significance level, usually set at 0.05 (5%), is the threshold for deciding whether the p-value is small enough to reject the null hypothesis.
2. If **p-value < α**, we reject H_0.
3. If **p-value ≥ α**, we fail to reject H_0.

3. Collect the Sample Data:

1. You gather the data from a sample, which is a smaller group taken from the entire population.

4. Calculate the Test Statistic:

1. Based on the sample data, you calculate a test statistic (like a **t-statistic** or **z-score**) that measures how much the sample data deviates from the null hypothesis.

5. Determine the p-value:

1. Use statistical methods or software (like Python) to calculate the p-value.

6. Make a Decision:

1. If the p-value is **less than the significance level (α)**, we reject the null hypothesis and conclude that there is enough evidence to support the alternative hypothesis.
2. If the p-value is **greater than or equal to α**, we fail to reject the null hypothesis, meaning there isn't enough evidence to support the alternative hypothesis.

A teacher claims that the average test score in her class is **70** ($H_0 : \mu = 70$).

You collect scores from **5 students**: [65, 70, 75, 80, 85].

You want to test if the data supports the claim ($H_1 : \mu \neq 70$).

1. State the Hypotheses

- Null Hypothesis (H_0): The average score is 70 ($\mu = 70$).
- Alternative Hypothesis (H_1): The average score is not 70 ($\mu \neq 70$).

2. Collect Data

- Sample data: [65, 70, 75, 80, 85].
- Sample mean ($\bar{x}$) = $\frac{65+70+75+80+85}{5} = 75$.
- Sample size (n) = 5.
- Sample standard deviation (s) = 7.91 (calculated as the spread of the data).

3. Calculate the t-Statistic

The t-statistic formula is:

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}}$$

Substitute the values:

- $\bar{x} = 75, \mu = 70, s = 7.91, n = 5,$

$$t = \frac{75 - 70}{\frac{7.91}{\sqrt{5}}} = \frac{5}{3.54} \approx 1.41$$

4. Determine the p-Value

- Degrees of freedom ($df = n - 1 = 4$).
- Use a t-distribution table or Python (`scipy.stats`) to find the p-value.

```
from scipy import stats
# Given data
t_stat = 1.41
df = 4
# Two-tailed p-value
p_value = stats.t.sf(abs(t_stat), df) * 2
print(p_value)
```

The p-value is approximately **0.230**.

5. Compare p-Value with Significance Level

- Significance level (α) = 0.05 (common threshold).
- p-value = 0.230.

Since $p > \alpha$, we **fail to reject the null hypothesis**.

6. Conclusion

- The data does not provide strong evidence to conclude that the average score is different from 70.
- Therefore, the teacher's claim ($H_0 : \mu = 70$) stands.

Hypothesis testing helps:

- Make decisions based on data.

- Avoid relying on assumptions.

- Assess claims, such as determining whether a new teaching method is effective

```
from scipy import stats

# Sample data: Heights of 10 students
sample_data = [150, 160, 155, 165, 170, 175, 160, 158, 162, 168]

# Population mean to test against
population_mean = 160

# Perform one-sample t-test
t_stat, p_value = stats.ttest_1samp(sample_data, population_mean)

# Set significance level
alpha = 0.05

# Check if we reject the null hypothesis
if p_value < alpha:
    print(f"Reject H₀: p-value = {p_value:.4f}")
else:
    print(f"Fail to Reject H₀: p-value = {p_value:.4f}")
```

Fail to Reject H_0: p-value = 0.5225

1. State the Hypotheses:

- Null Hypothesis (H_0): The average height of students is 160 cm.
- Alternative Hypothesis (H_1): The average height of students is not 160 cm.

2. Set the Significance Level (α):
- Let's choose **α = 0.05**, which means we are willing to accept a 5% chance of rejecting the null hypothesis when it is actually true.

3. Sample Data:
- We collect a sample of 10 students' heights: [150, 160, 155, 165, 170, 175, 160, 158, 162, 168]

4. Perform the Hypothesis Test:
- We use a **t-test** to compare the sample mean with the hypothesized population mean (160 cm).
- The test gives us a **p-value**.

5. Interpret the p-value:
- Suppose the p-value turns out to be **0.52**.

6. Decision:
- Since **0.52 > 0.05 (α)**, we fail to reject the null hypothesis. This means that there is not enough evidence to say that the average height is different from 160 cm. The sample data is consistent with the null hypothesis.

Key Functions

- `stats.ttest_1samp()` : Perform a one-sample t-test.
- `stats.ttest_ind()` : Conduct an independent two-sample t-test.
- `stats.ttest_rel()` : Perform a paired t-test.
- `stats.norm.ppf()` : Get the Z-score for a given confidence level.
- `stats.chi2_contingency()` : Perform a Chi-square test.

Excercises

- 1) Example: You want to select a simple **random sample** of 100 employees of Company X. You assign a number to every employee in the company database from 1 to 1000, and use a random number generator to select 100 numbers.

```
import random

population = 1000

data = range(population)

print(random.sample(data,100))
```

Sets the size of the population to 1000.

Creates a sequence of numbers from 0 to 999

Chooses 100 unique random elements from the data sequence

```python
import random

random.seed(120) #Prevents samples from changing each time code is run

fruits = ['mango', 'orange','apple','pawpaw','guava','strawberry','blackberry','cucumber','lemon']

fruit_sample = random.sample(fruits, k=3) # Three fruits are selected from the list of 9 fruits

print (fruit_sample)

#Output: ['lemon', 'pawpaw', 'orange']
```

2) Example: All employees of the company are listed in alphabetical order. From the first 10 numbers, you randomly select a starting point (**Systematic Sampling**): number 6. From number 6 onwards, every 10th person on the list is selected (6, 16, 26, 36, and so on), and you end up with a sample of 100 people.

```
population = 1000

step = 10

sample = [element for element in range(6, population, step)]

print (sample)
```

$$\text{Count} = \frac{\text{population} - \text{start}}{\text{step}} = \frac{1000 - 6}{10} = 99$$

creates a list of numbers:

- Starting at 6 (start=6).
- Ending before population (stop=1000).
- Incrementing by step=10.

3) Example: The company has offices in 10 cities across the country (all with roughly the same number of employees in similar roles). You don't have the capacity to travel to every office to collect your data, so you use random sampling to select 3 offices – these are your **Cluster Sampling.**

Cluster sampling also involves dividing the population into subgroups, but each subgroup should have similar characteristics to the whole sample. Instead of sampling individuals from each subgroup, you randomly select entire subgroups.

```
import numpy as np
import random

clusters = 5
pop_size = 100
sample_clusters = 2

# Assign cluster IDs (1 to 5) to 100 samples
cluster_ids = np.repeat(range(1, clusters + 1), pop_size // clusters)

# Randomly select the IDs of two clusters
cluster_to_select = random.sample(list(set(cluster_ids)), sample_clusters)

# Extract samples corresponding to the selected cluster IDs
indexes = [i for i, x in enumerate(cluster_ids) if x in cluster_to_select]

# Extract the sample values corresponding to the selected indices
cluster_associated_elements = [el for idx, el in enumerate(range(1, pop_size + 1)) if idx in indexes]

print("Selected Cluster IDs:", cluster_to_select)
print("Samples Associated with Selected Clusters:", cluster_associated_elements)
```

Reading Data

Reading Data from a Text File

Reading Data from a CSV File

.data files were developed as a means to store data. A lot of the times, data in this format is either placed in a **comma separated value** format or a **tab separated value** format. Along with that variation, the file may also be in text file format or in binary. In which case, we will be needing to access it in a different method.

```
# reading from the file
file = open("biscuits.data", "r")
file.read()
file.close()

# writing to the file
file = open("biscuits.data", "w")
file.write("Chocolate Chip")
file.close()
```

```
import pandas as pd
# reading csv files
data =  pd.read_csv('file.data', sep=",")
print(data)

# reading tsv files
data = pd.read_csv('otherfile.data', sep="\t")
print(data)
```

Tutorial: How to Easily Read Files in Python (Text, CSV, JSON)

https://www.dataquest.io/blog/read-file-python/

Reading Data from a Text File

```
# Find the sum of numbers stored in a file
def sum_data(filename):
    s = 0
❶   with open(filename) as f:
        for line in f:
❷           s = s + float(line)
    print('Sum of the numbers: {0}'.format(s))

if __name__ == '__main__':
    sum_data('mydata.txt')
```

Reading Data from a CSV File

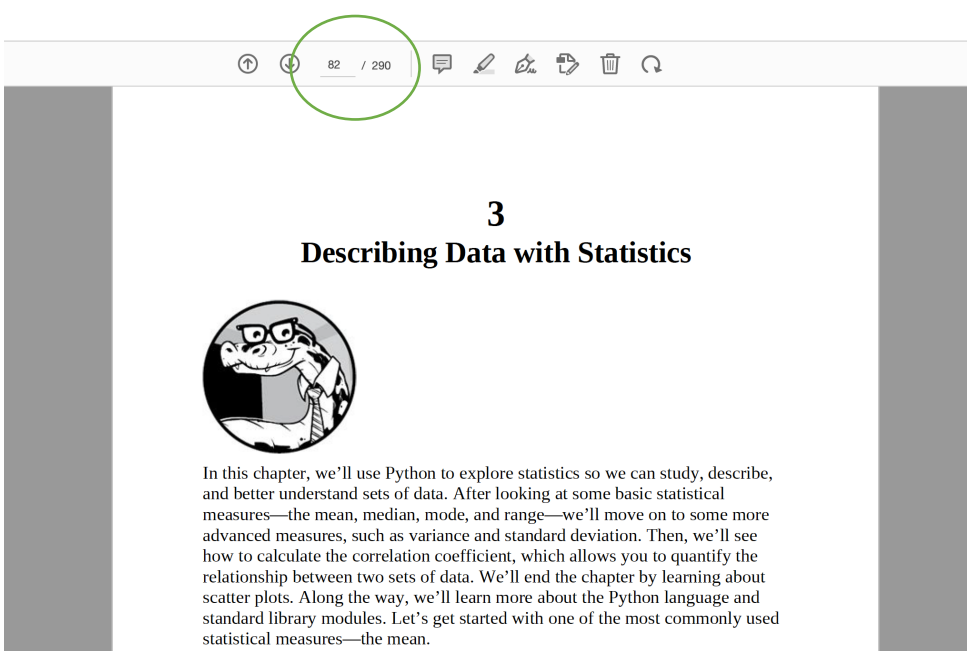
```
import csv
import matplotlib.pyplot as plt

def scatter_plot(x, y):
    plt.scatter(x, y)
    plt.xlabel('Number')
    plt.ylabel('Square')
    plt.show()

def read_csv(filename):

    numbers = []
    squared = []
    with open(filename) as f:
❶       reader = csv.reader(f)
        next(reader)
❷       for row in reader:
            numbers.append(int(row[0]))
            squared.append(int(row[1]))
        return numbers, squared

if __name__ == '__main__':
    numbers, squared = read_csv('numbers.csv')
    scatter_plot(numbers, squared)
```

Practical Exercise

Read the PDF file in google class.

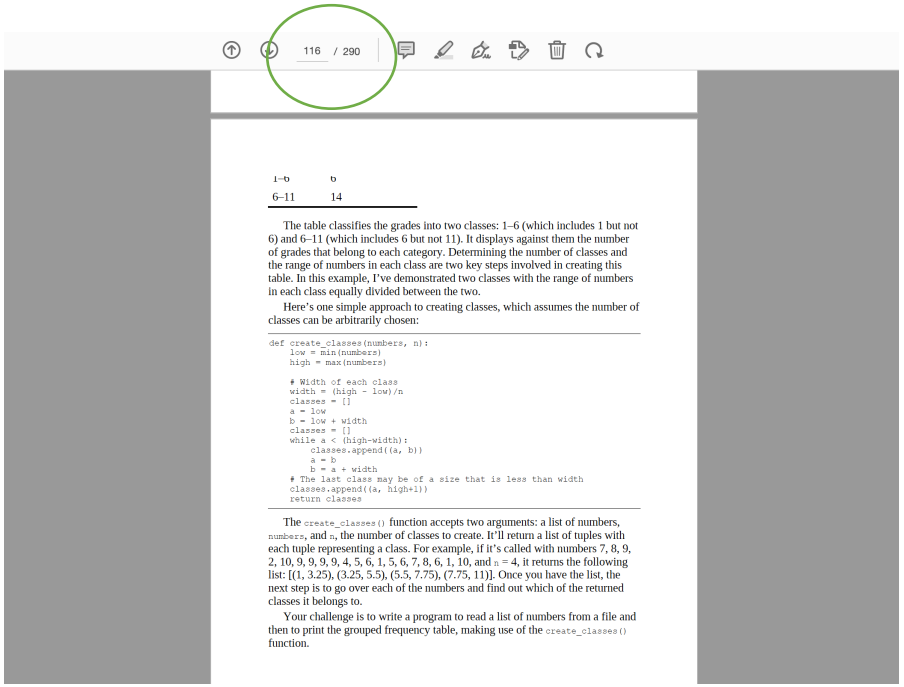
Students should read the following pages in detail

DOING MATH WITH PYTHON
Use Programming to Explore Algebra, Statistics, Calculus, and More!
by Amit Saha



3
Describing Data with Statistics

In this chapter, we'll use Python to explore statistics so we can study, describe, and better understand sets of data. After looking at some basic statistical measures—the mean, median, mode, and range—we'll move on to some more advanced measures, such as variance and standard deviation. Then, we'll see how to calculate the correlation coefficient, which allows you to quantify the relationship between two sets of data. We'll end the chapter by learning about scatter plots. Along the way, we'll learn more about the Python language and standard library modules. Let's get started with one of the most commonly used statistical measures—the mean.



1–6	6
6–11	14

The table classifies the grades into two classes: 1–6 (which includes 1 but not 6) and 6–11 (which includes 6 but not 11). It displays against them the number of grades that belong to each category. Determining the number of classes and the range of numbers in each class are two key steps involved in creating this table. In this example, I've demonstrated two classes with the range of numbers in each class equally divided between the two.

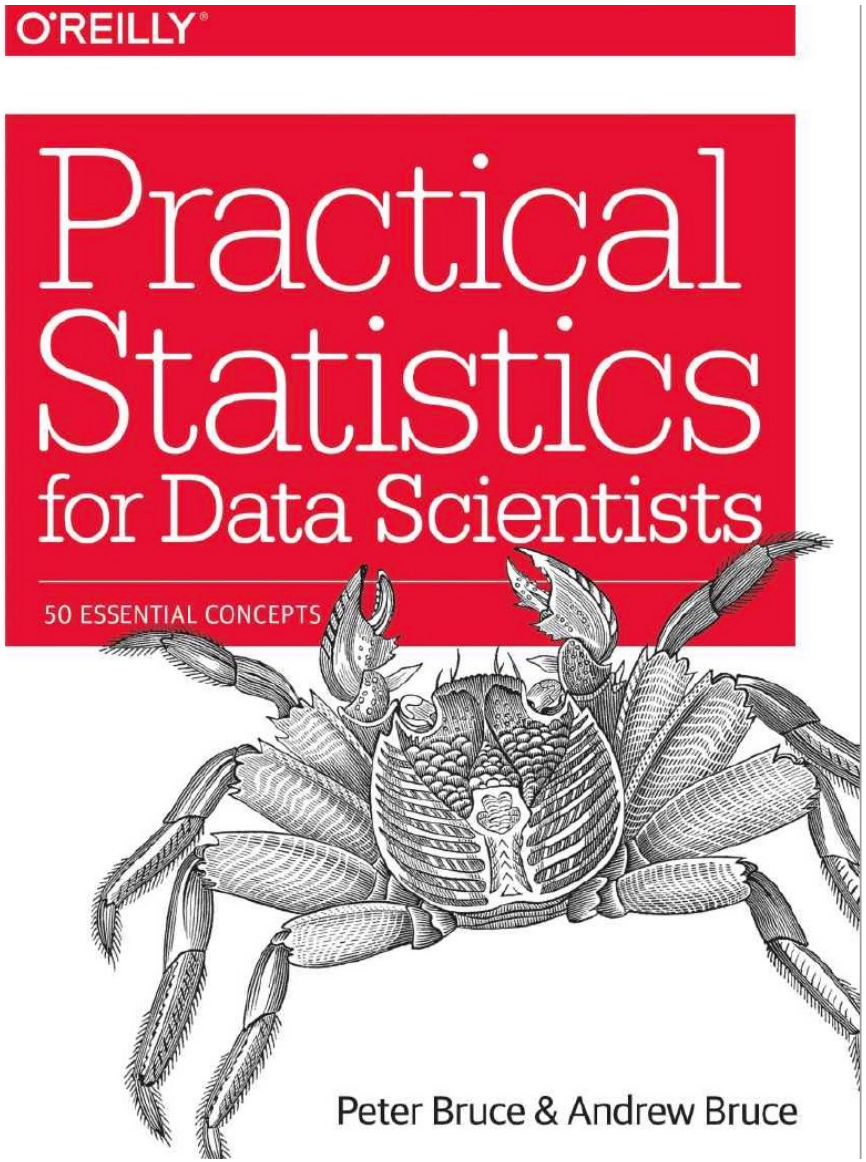
Here's one simple approach to creating classes, which assumes the number of classes can be arbitrarily chosen:

```
def create_classes(numbers, n):
    low = min(numbers)
    high = max(numbers)

    # Width of each class
    width = (high - low)/n
    classes = []
    a = low
    b = low + width
    classes = []
    while a < (high-width):
        classes.append((a, b))
        a = b
        b = a + width
    # The last class may be of a size that is less than width
    classes.append((a, high+1))
    return classes
```

The `create_classes()` function accepts two arguments: a list of numbers, `numbers`, and `n`, the number of classes to create. It'll return a list of tuples with each tuple representing a class. For example, if it's called with numbers 7, 8, 9, 2, 10, 9, 9, 9, 9, 4, 5, 6, 1, 5, 6, 7, 8, 6, 1, 10, and `n` = 4, it returns the following list: [(1, 3.25), (3.25, 5.5), (5.5, 7.75), (7.75, 11)]. Once you have the list, the next step is to go over each of the numbers and find out which of the returned classes it belongs to.

Your challenge is to write a program to read a list of numbers from a file and then to print the grouped frequency table, making use of the `create_classes()` function.

Recommended book for students, obtain extra knowledge about Statisitcs for data Science



What You Learned

End of Class 4