Applied Statistics - Notes - v0.7.0

260236

July 2025

Preface

Every theory section in these notes has been taken from two sources:

- The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. [2]

- An Introduction to Statistical Learning: with Applications in Python. [4]

- Applied Multivariate Statistical Analysis. [5]

- Course slides. [6]

About:

 GitHub repository

These notes are an unofficial resource and shouldn't replace the course material or any other book on applied statistics. It is not made for commercial purposes. I've made the following notes to help me improve my knowledge and maybe it can be helpful for everyone.

As I have highlighted, a student should choose the teacher's material or a book on the topic. These notes can only be a helpful material.

Contents

1   Business Data Analytics

1.1   Multivariate Descriptive Statistics

When we move from **analyzing** a single variable (univariate analysis) to **multiple variables at once**, we enter the realm of **Multivariate (MV) analysis**. A natural question arises: *Is multivariate analysis just a replication of univariate analysis across several variables?*

The answer is no, multivariate analysis introduces new and fundamental questions that cannot be answered by simply analyzing variables individually. The **core focus** shifts to **understanding how these variables interact with each other**. Specifically, we are concerned with the **dependence** and **correlation between variables**.

▤ Covariance: Measuring Joint Variability

To capture how two variables vary together, we use **Covariance**. The **Sample Covariance** between variables x_j and x_k is calculated as:

$$\text{cov}_{jk} = \text{Cov}\,(x_j, x_k) = s_{jk} = \frac{1}{n} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)(x_{ik} - \bar{x}_k) \tag{1}$$

- $s_{jk} = 0 \Rightarrow$ implies that there is **no linear relationship** between the two variables.

- $s_{jk} > 0 \Rightarrow$ suggests that **as one variable increases**, the **other tends to increase**.

- $s_{jk} < 0 \Rightarrow$ one **variable** tends to **decrease when the other increases**.

⚠ Covariance Is Not Standardized

The **value of covariance is not standardized**, it **depends on the units of measurement**, which makes comparisons difficult. For example:

- Suppose we're measuring

    - Height in centimeters
    - Weight in kilograms

- The covariance between height and weight will be expressed in *centimeter-kilogram* units.

Now imagine we convert height to meters. The covariance value changes, because now we're multiplying meters × kilograms instead of centimeters × kilograms. Even though the **relationship between height and weight hasn't changed**, the **numerical value of covariance does change due to this unit change. Because of unit dependency, it's hard to compare covariances between different variable pairs**. Finally, it is **hard to interpret the magnitude of covariance in any absolute sense** (e.g., is 50 a large covariance or small? It depends on the units!).

✅ Correlation: Standardized Covariance

To standardize covariance and **measure the strength of a linear relationship** on a scale between -1 and 1, we use the **Correlation** coefficient, defined as:

$$\text{cor}_{jk} = r_{jk} = \frac{s_{jk}}{\sqrt{s_{jj}}\sqrt{s_{kk}}} = \frac{\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)\left(x_{ik} - \bar{x}_k\right)}{\sqrt{\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2}\sqrt{\sum_{i=1}^{n}\left(x_{ik} - \bar{x}_k\right)^2}} \tag{2}$$

This formula divides the covariance by the product of the standard deviations of the two variables, giving a **unitless value**:

- $r = 0$: No linear correlation

- $r > 0$: Positive correlation (**both variables increase or decrease together**)

- $r < 0$: Negative correlation (**one increases while the other decreases**)

- $|r| = 1$: Perfect correlation (**exact linear dependence**)

Thus, correlation not only **reveals the direction of the relationship** but also its **strength**.
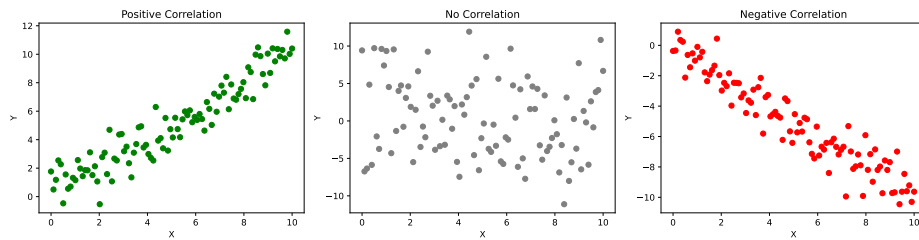
Figure 1: Direction of correlation: positive (left), none (center), negative (right).

▤ Describing MV Data: Vector and Matrices

When analyzing multivariate data:

- We compute the **vector of Sample Means**:

$$\bar{\mathbf{X}} = [\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_p] \tag{3}$$

- And the **symmetric and positive semidefinite** (eigenvalues are non-negative) **Variance-Covariance matrix S**, which **summarizes the covariances between all pairs of variables**:

$$S = \begin{bmatrix} s_{11} & \cdots & s_{1p} \\ \vdots & \ddots & \vdots \\ s_{p1} & \cdots & s_{pp} \end{bmatrix} \tag{4}$$

- Alternatively, we can use the **Correlation matrix R**, where **all diagonal elements are 1** (because each variable is perfectly correlated with itself) and **off-diagonal elements are correlation coefficients**:

$$R = \begin{bmatrix} 1 & \cdots & r_{1p} \\ \vdots & \ddots & \vdots \\ r_{p1} & \cdots & 1 \end{bmatrix} \tag{5}$$

📗 Scatterplots - Visualizing Variable Pairs

One of the most intuitive and widely used tools in multivariate analysis is the **2D Scatterplot**. Each plot shows how two variables relate to each other:

- ✔ Clusters or linear trends can indicate correlation or dependence.

- ✔ Scatterplots are **ideal for spotting positive, negative, or no correlation** visually.

However, scatterplots have a **limitation**: they **only** allow us to **analyze two variables at a time**. When dealing with many variables, the **number of possible pairings becomes large**, making it **difficult to read or interpret** the scatterplots individually. This is where quantitative measures (like correlation matrices) and higher-dimensional graphics come into play.

Figure 2: Scatterplot matrix of four variables. This scatterplot matrix displays all pairwise relationships among four variables:

- Diagonal plots (top-left to bottom-right): Histograms showing the distribution of each variable.

- Off-diagonal plots: 2D scatterplots illustrating the relationship between each pair of variables.

📗 Rotated Plots in 3D - Capturing Complexity

When dealing with **three variables**, we can **extend scatterplots into 3D space** using **Rotated plots**. These visualizations allow us to:

- ✔ **Explore interdependencies** among three variables at once.

- ✔ **Gain spatial insight** into how data points spread in three-dimensional space.

- ✔ Observe **complex patterns that are invisible in 2D**.

Yet again, as we move **beyond three variables**, visualizing becomes **impractical**, our brains cannot easily comprehend 4D or higher dimensions. Hence, dimensionality reduction techniques like PCA are often used alongside visualizations to make high-dimensional data "digestible".

Figure 3: A simple rotated plots in 3D.

▤ Star Plots - Shape-Based Comparison

Star plots offer a creative way to **represent multivariate data**:

- **Each variable is represented as a ray** (spoke) starting from a central point.

- The **length of each ray** corresponds to the **value** of that variable.

- When **rays are connected**, they form a "star-like shape" **unique** to each observation.

This method is **excellent for comparing patterns** between observations:

- ✔ **Similar shapes** suggest **similar data profiles**.

- ✔ **Differences** in shape can **quickly highlight outliers** or clusters.

However, star plots have limitations:

- ✖ They **do not quantify correlation**.

- ✖ The **direction** and **magnitude** of relationships between variables are **not explicit**.

- ✖ They are better for **visual pattern recognition** than for precise statistical analysis.

Figure 4: Star Plot (Radar Chart) of Multivariate Data.

📗 Chernoff Faces - Human-Centric Visualization

Chernoff faces [1] are an **innovative visualization method** where **multivariate data is represented as a human face**:

- Each **variable controls a facial feature** (e.g., mouth curvature, eye size, nose length).

- **People are naturally attuned to recognizing faces** and subtle differences in expressions.

- Hence, Chernoff faces **leverage human perception** for quickly comparing **multivariate observations**.

Despite being engaging, Chernoff faces also have **drawbacks**:

- ✖ They **do not provide numerical precision**.

- ✖ The **mapping of variables to facial features** can be **arbitrary**.

- ✖ They work best as a **qualitative summary tool** rather than for deep statistical inference.
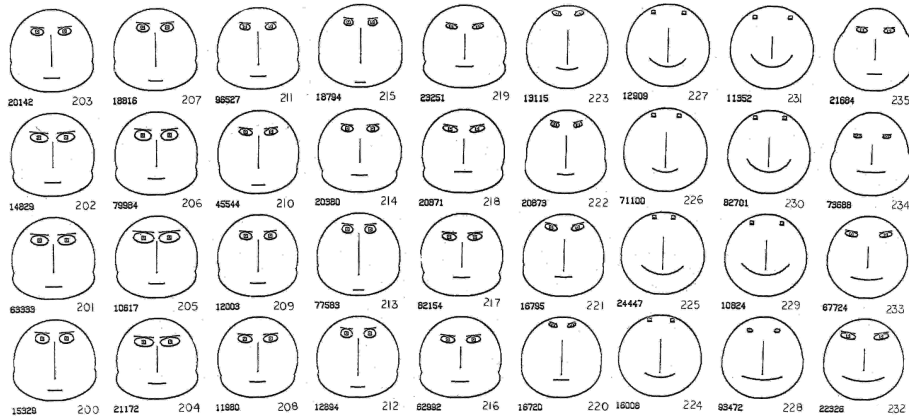
Figure 5: Some Chernoff faces. [1]

Graphic Type	Strengths	Limitations
Scatterplots	Clear view of pairwise relationships	Hard to scale beyond 2 variables
3D Plots	Visualizes 3-variable interaction	Limited to 3 dimensions, requires rotation
Star Plots	Quick shape-based comparison across variables	No quantification, poor at showing correlations
Chernoff Faces	Leverages facial perception for comparison	Subjective, lacks precision

Table 1: When and why to use graphics.

1.2   Dimensionality Reduction

⚠ The Challenge: Data in High Dimensions

In many real-world problems, we **collect multiple variables for each observation**. For example, in a medical study, a patient might be described by age, weight, blood pressure, and dozens of test results. This leads to **high-dimensional data**, where **each observation is a point in a complex, multi-dimensional space** (formally, a Euclidean space of dimension p).

The problem?   As the **number of variables** (p) **increases**, the **data becomes harder to visualize, interpret, and model**:

- ❗ Some **variables** might be **redundant** or **highly correlated**.

- ❗ **Computations** become more **expensive**.

- ❗ **Patterns become obscured** by the complexity.

⚑ Goal of Dimensionality Reduction

We want to **summarize the data using fewer variables**, say k derived variables (with $k < p$), that still **retain most of the information**. This process is a balancing act:

- **Clarity**: fewer variables make data easier to understand and visualize.

- **Risk of oversimplification**: reducing dimensions too much can cause loss of important information.

The key concept here is that in statistics, **information is variability**. The more variability we retain from the original data, the more information we preserve.

> **Example 1: Blood Cells**
>
> Imagine we measure thickness and diameter for a set of red blood cells:
>
> - Each cell = one observation with two variables.
>
> - We can represent this as a table (numbers) or as a 2D scatterplot.
>
> Now we ask ourselves: *Can we describe these cells using only one feature instead of two?*
> If we choose only diameter or only thickness, we'll lose detail:
>
> - Some cells will appear more similar than they really are.
>
> - We miss variability that distinguishes them.
>
> So, we seek a better single feature, one that captures the most variation possible from both thickness and diameter combined.

📈 The Statistical Insight: Maximize Variability

Rather than randomly picking a feature, we **analyze the directions along which the data varies the most**.

1. First, we find the **direction of maximum spread**.

2. Then, the **second most spread direction**, orthogonal to the first.

This is the essence of **Principal Component Analysis (PCA)**, a **dimensionality reduction technique that finds the best directions** (linear combinations of variables) **to project the data**, while **maximizing retained variability**.

📑 Dimensionality Reduction in Practice

Let's formalize the idea:

- We start with a **data matrix** X of shape $n \times p$ (n observations, p variables).

- The **goal** is to **obtain a new matrix** M of shape $n \times k$, with $k < p$, that **captures most of the variability**.

- The **difference** between X and M is **residual variation**, the **information lost**.

In summary, **Dimensionality Reduction** is about simplifying complexity: **transforming a large set of variables into a smaller**, more interpretable set **without losing the essence of the data**. It's central to data exploration, preprocessing, and modeling, especially when working with high-dimensional datasets.

1.3   Principal Component Analysis

✅ Why PCA?

Imagine we have a data set with **many variables**, such as measurements of people, products, or cities. Some **variables might be closely related** (like height and weight), **while others might carry similar information**. This creates two challenges:

1. It becomes **hard to analyze and interpret** the data.

2. **Redundancy** can lead to inefficiency and confusion.

Principal Component Analysis (PCA) helps solve this by providing a **simplified version of the dataset**, where we focus only on the **essential information**.

📗 The Main Idea

PCA works by **creating new variables**, called **principal components**, that are:

- **Combinations** of the original variables.

- **Ordered** so that the **first component captures the most variability** in the data.

- Each **subsequent component captures the next most variability**, but only from what's left over, and each **new component is uncorrelated with the previous** ones.

Imagine this like finding better "angles" or "perspectives" from which to view our data, ones that maximize how much we can see (i.e., variability) with as few perspectives as possible.

📗 Variability is Information

In statistics, **Variability**, **how much values change from one observation to the next**, is considered **information**. The more variability a component captures, the **more useful it is** in understanding the data. So, PCA's job is to **find the directions in which the data varies the most**, and to use those directions to summarize the dataset.

❓ How PCA Changes the Dataset

Let's say we start with p variables (like height, weight, age, income...). **PCA gives us p principal components**, but the *magic* is that **we usually only need the first few** to understand most of what's going on.

So instead of working with the full, original dataset, we now have a **simpler version**:

- We still have the **same number of observations**.

- But each observation is now described by **fewer, more informative variables** (components).

This is called a **low-dimensional representation** of the data.

📊 Scores and Loadings - The Ingredients and the Result

In this new system:

- The **Scores** tell us where **each observation lies along the new axes** (the principal components).

- The **Loadings** tell us **how the original variables contribute to each component**.

Think of it like cooking:

- **Original variables** = ingredients

- **Loadings** = recipe instructions

- **Principal components** = final dishes

- **Scores** = ratings of the dishes for each person (observation)

√x̄ Matrix Representation

Let's denote the data matrix as X, with n rows (observations) and p columns (variables). PCA is essentially about **transforming this matrix X into a new matrix**, where:

- The data is now described by **principal components** instead of the original variables.

- The **goal is to rotate and simplify** the data in a way that **emphasizes the most important directions** (maximum variability).

Two key matrices in PCA:

- **Loadings Matrix (V)**

  - This matrix contains the **weights** used to build the principal components.

- Each **column** in V corresponds to a **principal component**.
- Each **value** in V shows **how much each original variable contributes** to the component.

We can think of V as the recipe book: it tells us how to combine original variables to create the new components.

- **Scores Matrix (U)**

  - This matrix contains the **projections of the data** onto the principal components.
  - Each **row** in U represents an **observation in the new PCA space**.
  - These are called scores, they tell us **where each observation lands along the new axes** (PC1, PC2...).

  So U is the result: it shows **how our data looks in the new, simplified system**.

PCA can be **computed using Singular Value Decomposition (SVD)**:

$$X = U \cdot S \cdot V^T \tag{6}$$

PCA **factorizes the data matrix** X. Here's what each matrix means:

- X: Original data ($n \times p$)

- U: Scores matrix ($n \times p$), orthogonal (columns are independent)

- S: Diagonal matrix of **singular values** (related to the variance explained)

- V^T: Transposed loadings matrix ($p \times p$), also orthogonal.

But for understanding, focus on this **simpler form**:

$$\textbf{PCA result} = \text{Scores} = X \cdot V$$

We **multiply the data X by the loadings matrix V** to obtain the **scores**.

However, both U and V are **orthogonal matrices**, meaning:

- Their columns are perpendicular (**no redundancy**).

- Principal **components are uncorrelated**, each new component captures new, non-overlapping information.

This is mathematically elegant and practically useful because it **removes multicollinearity** and makes downstream **analysis simpler and more robust**.

❷ How Much Information Do We Keep?

Each principal component has a **percentage of variance explained**, this tells us how much of the original data's information it retains. Often, **the first 1 or 2 components explain so much that we can ignore the rest**.

In conclusion, **PCA helps us focus on what matters** in our data. It's like cleaning our glasses: everything becomes sharper, simpler, and more meaningful. We go from being overwhelmed by numbers to seeing clear patterns. It's a tool used everywhere, from finance to biology, marketing to engineering, whenever people need to make sense of complex data.

1.4   PCA Reference System

❷ Why Do We Talk About Projections in PCA?

When we say that PCA projects data, we mean it **transforms the data points by placing them onto new axes**, the principal components, that better represent the structure of variability.

❷ *But why are projections so important?* Because PCA has **two goals**:

1. **Maximize variance**: We want the **data to spread out as much as possible along the new axis**. This spread means we're capturing differences between observations.

2. **Minimize residuals**: We want to **minimize the error** when we approximate each point using the new axes. This is done by projecting each point perpendicularly onto the new axes, just like the shortest path between a point and a line.

This is why we "keep talking about projections": they allow us to **retain the most information** with **the least distortion**.

☰ Generalizing to More Dimensions

In real datasets, we often have **more than two variables**. PCA scales to p dimensions, and the idea of projection still applies:

- PC1: The **direction in p-dimensional space** where data varies most.

- PC2: The next direction, **perpendicular to the first**, that captures remaining variability.

- This continues until we have p principal components, each orthogonal to the others.

So **PCA rotates the entire dataset into a new reference system** where the data structure is easier to understand.

▤ A New Reference System

After PCA, our data now lives in a **new coordinate system**:

- The **original variables** (e.g., height, weight) are no longer the axes.

- Instead, the axes are **principal components**, which are **combinations of the original variables**.

This new reference system has two main features:

1. **PCA results are rotation invariant**

   ▤ *What does this means?* It means that if we rotate our dataset (e.g., by changing the coordinate system), PCA still finds the same underlying structure, the same principal components (relative to the data).

❓ ***Why?*** PCA depends only on the relationships between the data points, specifically: the variances and covariances between variables. These are not affected by rotations of the data in space. Mathematically, PCA extracts eigenvectors of the covariance matrix that are invariant under rotation with respect to relative directions.

2. **Principal Components are independent (uncorrelated)**

📄 ***What does this means?*** The principal components (PC1, PC2, ...) are uncorrelated with each other. Knowing the value of one PC tells us nothing about the value of the next.

❓ ***Why?*** Each new component is orthogonal to the previous one. And since the correlation is equal to the cosine of the angle between the variables ($\cos(90) = 0$), PAC ensures that the correlation between the PCs is zero.

✔ In high dimensions, **collinearity** often indicates redundant information between variables and can cause problems in the analysis. **PCA solves** this problem by giving us independent, uncorrelated variables.

Figure 6: We have a bunch of scattered red dots. We have drawn a black line through the middle of the cloud of dots in the direction where the dots are most scattered. This line is our first principal component (PC1). For each red point, we drop a perpendicular line onto the black line. Where it lands, we place a blue point. This is called projection. Furthermore, the longer the black line, the more red points are taken into account (thus maximizing variance); on the other hand, the blue points must be close to the red points (shorter the distance) to minimize residuals and lose less information.

Figure 7: By drawing two lines, PC1 and PC2, we obtain a new reference system. It is a rotated system. The axes are principal components, which are combinations of the original variables.

⚒ Principal Components: Linear Combinations

Let's now talk about **how principal components are constructed**. Each principal component is a **linear combination of the original variables**. This means:

$$\text{PC}_1 = \phi_{11} \cdot x_1 + \phi_{21} \cdot x_2 + \cdots + \phi_{p1} \cdot x_p$$

But in general:

$$\text{PC}_j = \phi_{1j} \cdot x_1 + \phi_{2j} \cdot x_2 + \cdots + \phi_{pj} \cdot x_p \tag{7}$$

Where:

- The ϕ-values (phi) are called **Loadings**.

- They are the **weights assigned to each original variable**.

- The **higher the loading**, the **more that variable contributes** to that principal component.

For **each observation** (e.g., a person, product), we now **compute their position in the new PCA space**:

$$z_{ij} = \phi_{1j} \cdot x_{i1} + \phi_{2j} \cdot x_{i2} + \cdots + \phi_{pj} \cdot x_{ip} \tag{8}$$

19

These values z_{ij} are called **Scores**. They tell us:

- Where each observation lies **along a principal component axis**.

- How much that **component contributes** to describing this observation.

So now, instead of describing each person with p original variables, we describe them with k scores, where $k < p$, but we still capture the essence of their data.

1.5   PCA as Optimization Problem

◎ **What Is PCA Trying to Achieve?**

PCA aims to find new variables (**principal components**) that are:

- **Linear combinations** of the original variables.

- Chosen so that the **first principal component** (PC1) captures the **maximum possible variance** in the data.

- **Second and further PCs** capture the **remaining variance**, while being uncorrelated with all previous ones.

In other words, PCA tries to find a new axis (direction) along which the data varies the most. This is the first principal component. Once this direction is found, each data point can be projected onto it to create a simplified representation of the data.

To find this direction (**PCA1**), **PCA solves a mathematical optimization problem**. This is because the **goal is to find the direction that maximizes the variance** of the data projected onto that direction (we want to maximize the variance because it captures the most variability, information, in the data).

❷ **How Is This Formulated as an Optimization Problem? (PC1)**

1. **Define a New Variable Z_1**. Let's construct a new variable Z_1 which is a **linear combination** of the original variables:

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + \cdots + \phi_{p1} X_p$$

Where:

- X_j is the j-th variable (all observations for that variable)
- ϕ_{j1} is the j-th weight (loading) assigned to the variable X_j.

The subscript 1 indicates that they belong to the first principal component.

2. **Objective - Maximize Variance of Z_1**. We want to maximize the variance to capture the most variability (information) in the data:

$$\text{Maximize} \quad \text{Var}\,(Z_1) = \text{Var}\,(\phi_1^T X)$$

Where:

- X is the **data matrix** $n \times p$
- ϕ_1 is the vector $p \times 1$ of **weights** (**loadings**) used to build Z_1. It is transposed to allow multiplication with the data matrix X.

More specifically:

$$\text{Var}\,(Z_1) = \frac{1}{n} \sum_{i=1}^{n} z_{i1}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} \cdot x_{ij} \right)^2$$

Where z_{i1} is the **score** (see page 19) of the i-th observation on the **first principal component** (e.g., z_{i2} score of observation i on PC2). In our case it is:

$$z_{i1} = \sum_{j=1}^{p} \phi_{j1} \cdot x_{ij}$$

3. **Constraint - Normalize the Loadings**. We need a **constraint** because we could scale the weights (loadings) infinitely. Therefore, we need **to fix the length** (norm) **of the loadings vector**.

$$\sum_{j=1}^{p} \phi_{j1}^2 = 1$$

Or, in vector notation:

$$||\phi_1||^2 = 1$$

The constraint says that the total energy or length of the loadings vector is fixed to 1 and PCA can only choose the direction of ϕ_1, not its size.

Now we can write the **full optimization problem for PC1**:

$$\text{Maximize} \quad \text{Var}\left(Z_1\right) \quad \text{subject to} \quad ||\phi_1||^2 = 1 \tag{9}$$

Specifically, the extended form:

$$\max_{\phi_{11},\ldots,\phi_{p1}} \quad \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j1} \cdot x_{ij} \right)^2 \quad \text{subject to} \quad \sum_{j=1}^{p} \phi_{j1}^2 = 1 \tag{10}$$

❓ **Okay, but how does PCA solve the optimization problem?**

In previous steps, we posed PCA as:

- Maximize the variance of $Z_1 = \phi_1^T X$

- Subject to $||\phi_1||^2 = 1$

The solution to this problem: the **optimal loadings vector** ϕ_1 (for PC1) **is the first eigenvector of the sample covariance matrix S**. Let's analyze this sentence to understand how and why.

❓ **Why Eigenvectors Help Solve This?** We need eigenvectors because PCA's optimization problem is mathematically equivalent to a linear algebra problem whose solution requires eigenvectors.

The **mathematical problem can also be written in quadratic form**[1]

$$\mathrm{Var}\,(Z_1) = \phi_1^T S \phi_1$$

Where S is the covariance matrix (page 7). It follows the exact pattern of the quadratic form, because ϕ_1 is a vector of loadings, S is the covariance matrix (square), and the whole expression evaluates to a scalar: the variance of Z_1.

In linear algebra, the **quadratic form problem is solved by eigenvectors**. Using the **Rayleigh Quotient Theorem**, we can obtain the goal of PCA. Given a symmetric matrix S (like the covariance matrix in PCA), the *maximum value* of the quadratic form:

$$\phi_1^T S \phi_1 \quad \text{subject to} \quad ||\phi_1||^2 = 1$$

Is achieved when ϕ_1 is the eigenvector corresponding to the largest eigenvalue of S. Therefore, the **solution must be the first eigenvector** (by the Rayleigh Quotient Theorem).

---

[1]A **Quadratic Form** is any expression that involves a vector, a matrix, and the same vector transposed. The general structure is:

$$Q(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} \tag{11}$$

Where $\mathbf{x}$ is a vector, $\mathbf{A}$ is a square matrix, and the result is a scalar.

❷ Optimization Problem for the Second Principal Component (PC2)

The following steps allow to obtain the PC2. We avoid a long explanation here, because we made the same logical steps of the optimization problem for PC1.

1. **Define a New Variable Z_2.** Just like with PC1, we define PC2 as a **linear combination** of the original variables:

$$Z_2 = \phi_{12}X_1 + \phi_{22}X_2 + \cdots + \phi_{p2}X_p$$

   For a specific observation i:

$$z_{i2} = \sum_{j=1}^{p} \phi_{j2} \cdot x_{ij}$$

   Where: $\phi_2 = [\phi_{12}, \ldots, \phi_{p2}]^T$ is loadings vector for PC2. Here the **goal is to find the best ϕ_2 to create PC2**.

2. **Objective - Maximize Variance of Z_2.** Just like PC1, we want to **maximize variance**:

$$\max_{\phi_2} \quad \mathrm{Var}\,(Z_2) = \frac{1}{n}\sum_{i=1}^{n} z_{i2}^2$$

3. **Constraints (there are two now)**

   (a) **First Constraint: Normalize the Loadings.** We must prevent arbitrary scaling of the weights:

$$\sum_{j=1}^{p} \phi_{j2}^2 = 1 \quad \text{or} \quad \|\boldsymbol{\phi}_2\|^2 = 1$$

   (b) **Second Constraint: Ensure PC2 is Uncorrelated with PC1.** This is new compared to PC1. Here we want the covariance between PC1 and PC2 to be zero; in other words, we're saying ***we don't want these two results to be correlated***.

$$\mathrm{Cov}(Z_1, Z_2) = 0$$

   Mathematically, this is equivalent to saying:

$$\phi_1^T \phi_2 = 0 \qquad \text{(orthogonality)}$$

   If **two directions are orthogonal**, their **projections** (scores) are **uncorrelated**.

Finally, we can write the **full optimization problem for PC2**:

$$\max_{\phi_{12},\ldots,\phi_{p2}} \quad \frac{1}{n}\sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{j2}x_{ij}\right)^2 \tag{12}$$

Subject to:

$$\sum_{j=1}^{p} \phi_{j2}^2 = 1 \quad \text{and} \quad \sum_{j=1}^{p} \phi_{j1} \cdot \phi_{j2} = 0 \tag{13}$$

❓ **And how to solve the PC2 optimization problem?** The solution to this constrained optimization problem is similar to the PC1:

$$\phi_2 = \mathbf{e}_2 \tag{14}$$

Where:

- $\mathbf{e}_2$ equal to the **second eigenvector** of the **covariance matrix** S.

- This eigenvector corresponds to the **second largest eigenvalue** λ_2.

1.6 Proportion of Variance Explained (PVE)

In the previous sections, we explained how to compute PC1, PC2, etc. as new variables. Each PC is a linear combination of the original variables, and each maximizes the variance under orthogonality constraints. But now the question becomes: "*okay, we computed these new components... but how useful are they?*".

In PCA, variance equals information. So now our **goal** is: "how much **information (variance) is explained by PC1**? And by **PC2**? And by **PC3**? And so on". This is where **Proportion of Variance Explained (PVE)** comes in.

We know that each principal component explains some variance:

- PC1 captures the **largest possible amount** of variance.

- PC2 captures the **next largest amount**, and so on.

- All PCs **together** explain **100% of the variance** in the data (no information is lost if we keep all of them).

If the data matrix X is centered (i.e., variables have mean 0), then the **total variance in the data** is:

$$\text{Var}_{\text{TOT}} = \sum_{j=1}^{p} \text{Var}(X_j) = \sum_{j=1}^{p} \left(\frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 \right) \tag{15}$$

This is just the **sum of the variances** of the original variables.

To know the **variance** at the step k, so for the k-**th principal component**, we generalize:

$$\text{Var}_k = \frac{1}{n} \sum_{i=1}^{n} z_{ik}^2 \tag{16}$$

Where z_{ij} is the **score of observation** i on component k. In other words, it is the **spread of data along PC**k.

Since the **PVE** is a proportion, the equation is pretty obvious:

$$\text{PVE}_k = \frac{\text{Var}_k}{\text{Var}_{\text{TOT}}} \tag{17}$$

It is the **percentage of total variance** explained by PCk. This tells us how informative PCk is, and whether it's worth keeping or can be discarded.

In practice, we **often keep only the first few PCs**. For example, if we find that $PC1 + PC2$ explain 90% of the variance, we can keep only those two (dimensionality reduction).

1.7 Covariance vs. Correlation in PCA

In the previous sections, we discussed what PCA is and how we can use it. We know that PCA is based on the covariance matrix, that each principal component explains a portion of the total variance, and that PCA depends on the variance of the data. But "**what happens when the variables are on completely different scales?**" (Variable 1 is height, variable 2 is euros, variable 3 is just a count). If we apply PCA to the covariance matrix, the **variable with the largest variance will dominate the result**, even if it's not more important!

More formally, **PCA is based on the spread** (variance) **and co-movement** (covariance) **between variables**. But this depends heavily on the units of measurement and scales of the variables. And the problem is, should **we use the covariance matrix or the correlation matrix** to perform PCA? The answer depends on the scales of our variables.

⚠ Covariance Matrix: use with caution

The **covariance matrix measures how variables change together**, in ==absolute units==.

  ✔ This is fine only if **all variables are measured on comparable scales** (e.g., all in centimeters, or all in euros).

  ✘ If one variable has **much larger variance**, it will **dominate the PCA**, even if it's not the most important feature.

For example, suppose height (meters) has a variance of 0.01 and income (euros) has a variance of $10'000$. Income will control the direction of PC1 simply because its variance is greater, not because it's more meaningful.

✅ Solution: Standardize the Data

To prevent PCA from being biased toward large-scale variables, we **standardize** each variable:

$$X'_{ij} = \frac{X_{ij} - \bar{X}_j}{\text{SD}_j} \tag{18}$$

Where:

  • $\bar{x}_j$ is the mean of variable j.

  • SD_j is the standard deviation of variable j.

This standardization ensures that the **mean is zero** and the **standard deviation is one**. Now **all variables are on the same scale** and PCA treats them equally.

✅ **Alternative: use the correlation matrix instead of the covariance matrix**

Another way to achieve this is to run PCA on the **correlation matrix**, which is simply the **covariance matrix of the standardized variables**. This work because:

$$\text{Corr}(X_j, X_k) = \frac{\text{Cov}(X_j, X_k)}{\text{SD}\,(X_j) \cdot \text{SD}\,(X_k)} \tag{19}$$

So if we **standardize the variables first, the covariance becomes correlation**.

Case	Matrix to use	Why
All variables are on the same scale	Covariance matrix	Keeps units and variances as they are
Variables have different units/scales	Correlation matrix	Avoids bias toward large-variance variables
We standardized the variables	Covariance = correlation	Because standardization equalizes them

Table 2: When to use Covariance or Correlation.

1.8   PCA via SVD - Computational Aspects

Instead of computing PCA by:

1. Building the covariance matrix $S = \dfrac{1}{n-1}X^T X$

2. Diagonalizing S to get eigenvectors and eigenvalues

We can instead **perform PCA directly using Singular Value Decomposition (SVD)** on the **centered data matrix** X itself.

▤ Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) method is a factorization of a matrix into three other matrices. If X is the centered data matrix (size $n \times p$), then:

$$X = U\Sigma V^T \tag{20}$$

Where:

- **U**: matrix of **left singular vectors** (related to scores). It is orthogonal and $n \times n$.

- **Σ**: diagonal matrix of **singular values** $\Sigma_1, \Sigma_2, \ldots$

    - Non-negative real numbers on the diagonal (singular values)

    - Singular values sorted in descending order (from largest to smallest)

    - Number of eigenvalues guaranteed by the minimum between number of columns and number of rows: $\min(n, p)$.

- **V**: matrix of **right singular vectors**. It is orthogonal and $p \times p$. They are the main directions, so the loadings of PCA.

The eigenvalues of the covariance matrix can be calculated from the singular values Σ_i:

$$\lambda_i = \frac{\Sigma_i^2}{n-1}$$

This is because if we plug the SVD of X into this formula, the eigen decomposition of S naturally follows.

Concept	Meaning/Role in PCA
$\mathbf{X} = \mathbf{U\Sigma V}^T$	SVD decomposition of centered data matrix
V	Principal directions (same as eigenvectors of covariance)
Σ_i	Singular values
$\lambda_i = \dfrac{\Sigma_i^2}{n-1}$	Variance explained by each principal component

Table 3: Summary table of SVD for PCA.

✔ Advantages

    ✔ Numerically more stable: works better when the data matrix is large or ill-conditioned.

    ✔ No need to compute and store the covariance matrix: we can work directly on X.

    ✔ More efficient when p is large (many variables), especially in high-dimensional problems.

2   Clustering Methods

2.1   Introduction

In the previous section, we talked about PCA for compression or visualization. However, this topic falls under a type of technique called unsupervised learning.

- **Supervised Learning** is a type of machine learning where the algorithm is trained on a labeled dataset, meaning **each** training **example includes both the input data and the correct output**.

  The **goal** is to learn a function that maps inputs to outputs, in order to make predictions on new, unseen data. Typical tasks include:

  - Classification (e.g., spam detection, image recognition)
  - Regression (e.g., predicting house prices or credit scores)

- **Unsupervised Learning** is a type of machine learning where the algorithm is given **only input data without any labeled output**.

  The **objective** is to identify patterns, structures or groupings within the data. Common applications include:

  - Clustering (this section)
  - Dimensionality Reduction (e.g., PCA for compression or visualization, section 1.2, page 12)

In essence, while supervised learning is about *prediction*, unsupervised learning is about *discovery*.

❷ What is Clustering?

Clustering is the ==process of grouping a set of objects== in such a way that **objects within the same group (cluster)** are **more similar** to each other than to those in other groups.

- No "true" labels are provided (unsupervised learning), the objective is to **uncover structure**.

- Often used in exploratory data analysis.

- The notion of "similarity" is fundamental and context-dependent (commonly based on **distance measures**).

The basic idea of clustering is:

1. **Minimize intra-cluster distances**: member of the **same cluster** should be **close to each other**.

2. **Maximize inter-cluster distances**: **different clusters** should be **well separated**.

> ### Example 1: Energy Consumption in Milan
>
> A practical example:
>
> - Consider n users, each described by their energy consumption across p time slots.
>
> - The task is to identify **groups of users with similar consumption patterns**, potentially revealing roles like *residences vs offices* or *daytime vs nighttime* usage.
>
> - Since there is no predefined label, this is a classic unsupervised learning task.
>
> This example encapsulates the **essence of clustering**, discovering structure in data that wasn't explicitly labeled.

❷ How Many Clusters?

A fundamental **challenge** in clustering is **deciding the number of clusters** (k). Visualizations show that data can often be reasonable clustered in 2, 4, or 6 groups, depending on the chosen definition or similarity.

The **clustering result depends on the similarity measure** (e.g., Euclidean distance), and the notion of a "true" cluster is often ambiguous. This ambiguity is a central theme in unsupervised learning: since there's no ground truth, **evaluating the "correctness" of a clustering is inherently difficult**.

≡ Types of Clustering

There are two main paradigms:

1. **Hierarchical Clustering**: builds a tree of clusters. Two main strategies:

   - **Agglomerative**: start from individual points and merge them.
   - **Divisive**: start from the whole dataset and split it recursively/

2. **Partitional Clustering** (also called flat clustering): divides data into k non-overlapping groups. Each data point belongs to exactly one cluster.

Hierarchical methods are more interpretable, while partitional methods (e.g., k-means) are often faster and scalable.

2.2   Defining Similarity in Clustering

At the heart of clustering lies a simple yet crucial idea: "objects in the same cluster should be **more similar** to each other than to objects in other clusters". But how do we define "*similar*"? Clustering methods don't work in a vacuum, they **need a distance** (or similarity) **function to determine how close two data points are**. This choice directly affects the clustering result.

☰ Types of Distance Metrics

Given two data points $x = (x_1, x_2, \ldots, x_p)$ and $y = (y_1, y_2, \ldots, y_p)$, the distance between them can be computed in different ways, depending on the nature of the data and the goals of the analysis.

- **Euclidean Distance**

$$d_E\left(x, y\right) = \sqrt{\sum_{i=1}^{p} \left(x_i - y_i\right)^2} \tag{21}$$

  - Measures straight-line distance.
  - Works well when **features are on the same scale**.
  - Variants: squared euclidean and standardized euclidean (normalize variables before computing).

- **Manhattan Distance**

$$d_M\left(x, y\right) = \sum_{i=1}^{p} |x_i - y_i| \tag{22}$$

  Also called **L1 norm**, measures **grid-like distance** (think of navigating city blocks).

- **Chebyshev Distance**

$$d_C\left(x, y\right) = \max_i |x_i - y_i| \tag{23}$$

  Takes the **maximum absolute difference** across dimensions. Sensitive to the **worst-case** coordinate difference.

- **Cosine Similarity (Angle-based)**

$$\text{similarity}\left(x, y\right) = \cos\left(\theta\right) \tag{24}$$

  Based on the **angle between vectors**. The smaller the angle, the more similar the vectors. Common in **text mining** and **high-dimensional sparse data**.

- **Correlation-Based Distance**

$$d_R\left(x, y\right) = 1 - \text{corr}\left(x, y\right) \tag{25}$$

  Measures **shape similarity** and ignores magnitude; focuses on patter of variation.

- **Mahalanobis Distance**

$$d_M\left(x,y\right) = \sqrt{\left(x-y\right)^T \Sigma^{-1}\left(x-y\right)} \tag{26}$$

Accounts for **correlation between variables**. Useful when features have **different variances** or **covariances**.

- **Minkowski Distance**

$$d_p\left(x,y\right) = \left(\sum_{i=1}^{p}\left|x_i - y_i\right|^p\right)^{\frac{1}{p}} \tag{27}$$

General formula that generalizes Euclidean and Manhattan. If $p = 1$ is Manhattan, if $p = 2$ is Euclidean.

❷ Why Distance Matters

Clustering doesn't "know" what matter, we tell it via distance.

- The **choice of distance metric** can lead to **entirely different clusters**.

- Different applications require different definitions of similarity. For example, in customer segmentation, cosine similarity may be more appropriate than Euclidean distance, because we care about *purchasing trends*, not *absolute values*.

So instead of asking "*what's the best distance?*", ask: "***what kind of similarity is meaningful in our application?***"

2.3   Clustering Validity

Clustering can produce a result, a partition of our data, but **how do we know if it's good**? Unlike supervised learning, clustering lacks ground truth, so evaluating the quality of clustering is challenging. There are three main families of evaluation metrics, each answering a different kind of question: External Metrics, Internal Metrics, Relative Metrics.

2.3.1   External Metrics

External metrics assess the quality of a clustering by **comparing it to known class labels**. They answer the question: "how similar is the clustering result to the actual classification?". This is **possible only when ground truth labels[2] are available**, which is often the case in benchmarking or simulated data. However, these metrics are **not available in real-world applications where true labels are unknown**.

We usually **compare clusters** to classes using a **contingency table**:

	Class 1	Class 2	Class 3	Total
Cluster 1	m_{11}	m_{12}	m_{13}	m_1
Cluster 2	m_{21}	m_{22}	m_{23}	m_2
Cluster 3	m_{31}	m_{32}	m_{33}	m_3
Total	c_1	c_2	c_3	n

- m_{ij} number of points from class j assigned to cluster i

- m_i total points in cluster i

- c_j total points in class j

- n total number of points

From this, we compute **frequencies**:

$$p_{ij} = \frac{m_{ij}}{m_i} \tag{28}$$

---

[2]**Ground Truth Labels** are the true, correct categories or values assigned to data points. They represent the known answer. In clustering, ground truth refers to the real classification or grouping of our data, which is often manually annotated, observed from reality, or known from the context.

🟩 Main External Metrics

- **Entropy Metrics**. Measures **class diversity within each cluster**.

    - A **pure cluster** (all elements from the same class) has entropy 0.
    - A **mixed cluster** hsa higher entropy (maximum when classes are equally mixed).

  The formula for **cluster** i:

  $$e_i = -\sum_{j=1}^{L} p_{ij} \log (p_{ij}) \tag{29}$$

  Overall clustering entropy:

  $$e = \sum_{i=1}^{K} \frac{m_i}{n} e_i \tag{30}$$

  Entropy decreases with better alignment. The **ideal value is zero**.

- **Purity Metrics**. Measures the **proportion of the dominant class in each cluster**. Like a "best guess" correctness. The formula for **cluster** i:

  $$p_i = \max_{j} (p_{ij}) \tag{31}$$

  Overall clustering purity:

  $$\text{purity} = \sum_{i=1}^{K} \frac{m_i}{n} p_i \tag{32}$$

  Purity increases with better alignment. The **ideal value is one**.

- **Precision Metrics** and **Recall Metrics**. These are adapted from classification metrics:

    - Precision (for cluster i, class j):

      $$\text{Prec}(i, j) = \frac{m_{ij}}{m_i} \tag{33}$$

      Among the points in cluster i, **how many truly belong to class** j?
    - Recall (for cluster i, class j):

      $$\text{Rec}(i, j) = \frac{m_{ij}}{c_j} \tag{34}$$

      Among all points from class j, **how many are captured by cluster** i?

- **F-Measure Metrics**. Combines precision and recall into a **single score** using the harmonic mean.

  $$F(i, j) = \frac{2 \cdot \text{Prec}(i, j) \cdot \text{Rec}(i, j)}{\text{Prec}(i, j) + \text{Rec}(i, j)} \tag{35}$$

  We can aggregate these across all clusters to get a **global F-score** for the clustering.

Metric	Ideal	Interprets as...
Entropy	0	Purity of cluster (lower is better)
Purity	1	Dominance of single class (higher = better)
Precision	1	Cluster doesn't mix in wrong classes
Recall	1	Class is fully captured by a cluster
F-measure	1	Balanced precision and recall

Table 4: Summary External Metrics.

2.3.2 Internal Metrics

When class labels (ground truth) are not available, which is the most common case in unsupervised learning, we need **Internal Metrics** to assess:

- How **cohesive** each cluster is (tightness of points within clusters).

- How **well-separated the clusters are from each other**.

These metrics measure **structural quality** based on distances between two points.

- **Sum of Squared Errors (SSE)**. Also called **Within-Cluster Sum of Squares (WCSS)**.

  ❓ **What is measures**
    * **How far** the **points** in each cluster are from their **cluster center** (or medoid)
    * **Lower SSE means tighter clusters**[3]

  📑 **Formula**

  $$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - c_i||^2 \tag{36}$$

  Where:

    * C_i is cluster i
    * c_i is the center of cluster i

  Used to evaluate compactness: a good clustering should have a small SSE.

---

[3]A tight cluster means that the data points inside the cluster are close to each other. They are packed together, not scattered.

- **Elbow Method**

  - ❷ **What is measures**

    - ∗ Used with SSE to find the **optimal number of clusters** K.

  - ✂ **Procedure**

    - ∗ Run clustering for various values of K (e.g., 1 to 30)
    - ∗ Plot $SSE\,(K)$ vs K.
    - ∗ Find the "elbow" point: the value of K where the SSE stops decreasing sharply.

    In other words, adding more clusters beyond the elbow gives **diminishing returns**.

- **Silhouette Coefficient**

  - ❷ **What is measures**

    - ∗ **Combines cohesion and separation** into a single score for each point.
    - ∗ For each data point:
      - · a: average distance to points in **same cluster**
      - · b: average distance to points in **nearest other cluster**

  - 📕 **Formula**
    $$s = \frac{b - a}{\max\,(a, b)} \qquad s \in [-1, 1] \tag{37}$$

    - ✔ +1 well clustered
    - ∗ 0 on the border
    - ✘ −1 misclassified

    The **average silhouette score** across all points is **used to evaluate the whole clustering**.

- **Between-cluster Sum of Squares (BSS)**. Measures **cluster separation**:
  $$BSS = \sum_i m_i \, ||c_i - \bar{c}||^2 \tag{38}$$

  Where:

  - m_i: size of cluster i
  - c_i: center of cluster i
  - $\bar{c}$: global centroid

  A **good clustering** has **low WCSS** (or SSE) and **high BSS**.

⚠ Limitation of Internal Metrics

Most clustering algorithms **don't explicitly optimize internal metrics**. As a result, the **best-looking clustering under one metric may not be optimal in another**. For example, K-Means minimizes WCSS but may produce poor separation between clusters.

We can **design custom clustering algorithms that directly optimize an internal metric**, but this is <mark>not always practical or generalizable</mark>. A famous quote from Jain and Dubes (1988): "The validation of clustering structures is the most difficult and frustrating part of cluster analysis. Without a strong effort in this direction, clustering remains a black art accessible only to those true believers who have experience and great courage.". This reflects **how tricky and interpretation-heavy clustering evaluation can be**.

2.4 Hierarchical Clustering

Hierarchical Clustering is a **method of clustering that builds a hierarchy of clusters**, either by **merging** smaller clusters into larger ones (bottom-up) or **splitting** a large cluster into smaller ones (top-down).

- **Agglomerative Clustering (Bottom-Up)**

  1. Starts with **each data point as its own cluster**
  2. At each step:
     (a) Find the **two closest clusters**
     (b) **Merge** them into one
     (c) **Repeat** until there's one single cluster or a predefined number k

  This is the most common form of hierarchical clustering.

- **Divisive Clustering (Top-Down)**

  1. Starts with **all points in one large cluster**
  2. At each step:
     (a) **Split** one cluster into two
     (b) **Continue recursively** until each point is in its own cluster (or until k is reached)

  This method is less commonly used due to higher complexity.

✂ Implementation Agglomerative Hierarchical Clustering

Hierarchical clustering works with a **distance matrix**, which **contains the distance between every pair of observations**. We don't need to specify k beforehand, but we need a **stopping criterion** (e.g., stop when there are k clusters, or use a threshold on distance).

1. Compute the **proximity matrix** (distances between all points)

2. Let each point be its **own cluster**

3. **Repeat**:
   (a) Find and **merge the two closest clusters**
   (b) **Update** the distance matrix

4. **Stop** when only one cluster remains (or another stopping condition is met)

The behavior of step 3, how we defined "closest clusters", depends on the **linkage method**.

🔗 Linkage Methods: How we Merge Clusters

The **linkage method** defines the distance between two clusters, based on the distances between points in those clusters.

1. **Single Linkage Method**

   ❓ **Distance**. It is the **minimum distance** between any two points (nearest neighbors)

   ✅ Can handle **non-globular shapes**.

   ⚠️ Sensitive to **noise and chaining effects**.

2. **Complete Linkage Method**

   ❓ **Distance**. It is the **maximum distance** between any two points (furthest points).

   ✅ Less sensitive to noise, but biased toward **spherical or compact clusters** (we mean that the method tends to favor or naturally produces clusters that have a specific shape or property).

3. **Average Linkage Method**

   ❓ **Distance**. It is the **average distance** between all points pairs.

   ✅ Uses **centroids** as reference and offers a **balance** between single and complete.

✅ Pros

✔ Does not require predefining the number of clusters k.

✔ **Produces a dendrogram**[4], a tree-like diagram showing how clusters are formed.

✔ Can reveal structure at **multiple levels of granularity**.

❌ Limitations and Challenges

✖ **Computational complexity**: distance matrix is $O(n^2)$, problematic for large datasets.

✖ Once a merge/split is made, it **cannot be undone**.

✖ There's **no direct optimization** of a global objective (unlike k-means minimizing SSE) .

✖ **Results depend heavily on the linkage method used**, can lead to very different clusterings.

---

[4]The dendrogram is a tree where the leaves are individual data points and branches are merges between clusters. To choose k, "cut" the dendrogram at the level where the structure is stable (e.g., large vertical gaps suggest good splits).

2.5 K-Means

2.5.1 Introduction

K-Means is one of the most popular **clustering method**, specifically a type of **partitional clustering** (page 32). It's used to **group data into distinct clusters based on similarity**.

K-Means divides a dataset into K groups, where:

- Each group is called a **cluster**.

- **Each cluster** has a **center** called **centroid**.

- Every data point belongs to the cluster with the **nearest centroid**.

❓ How does it work?

The K-means method can be divided into **five steps**:

1. **Choose K** (the number of clusters).

2. **Randomly assign** each data point to a cluster.

3. **Compute the centroid** of each cluster (the mean of all its points).

4. **Reassign** each **point to the cluster** with the closest centroid.

5. **Repeat** steps 3 and 4 **until** the points **no longer change** clusters (convergence criterion).

The algorithm tries to **minimize the distance** between each point and the center of its cluster. This means we want the **clusters to be as compact** and **well-separated as possible**.

Note that this is only a simple, quick introduction to K-Means clustering. In the following chapters, we will delve deeply into each topic.

2.5.2   Definition

K-Means is a **unsupervised learning algorithm** used to partition a dataset into K **distinct non-overlapping groups** called **clusters**, where each observation belongs to the cluster with the nearest mean (called the **centroid**).

◎ Main Goal

The goal of K-Means is to find a partition of the data that **minimizes the total within-cluster variation**, usually measured as the **sum of squared Euclidean distances** between each point and the centroid of its assigned cluster.

In other words, the K-means algorithm aims to **partition a dataset of n observations into K distinct clusters**:

- Each observation belongs to **one and only one** cluster.

- Each cluster has a **centroid**, i.e., a central point.

- Each point is assigned to the cluster with the **nearest centroid**.

- The goal is to **minimize the total distance** (squared Euclidean distance) from each point to its cluster's centroid.

We can express the previous goals in raw mathematical terms as follows:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{39}$$

Where:

- C_k is the **set of points** in the k-th cluster. Let $C_1, C_2, \ldots, C_K$ be the **index sets** of the clusters; these must form a **partition** of the dataset:

    - $C_1 \cup C_2 \cup \cdots \cup C_K = \{1, 2, \ldots, n\}$, every observation is included in some cluster.
    - $C_i \cap C_j = \emptyset$ for $i \neq j$, no observation can belong to two clusters at once.

    So, if the index $i \in C_k$, that means the i-th observation is assigned to the k-th cluster.

- $\boldsymbol{\mu}_k$ is the **centroid** (mean vector) of cluster C_k.

- $\|\mathbf{x}_i - \boldsymbol{\mu}_k\| = \|\cdot\|$ denotes **Euclidean distance**. It is the quantity that K-Means tries to minimize over all clusters.

Concept	Meaning
Clustering type	Partitional (flat).
Centroid	Mean of points in a cluster.
Assignment rule	Assign each point to the **closest centroid**.
K **(number of clusters)**	Must be **chosen in advance**.
Objective function	Minimize total within-cluster distance (e.g., sum of squared distances).
Partition conditions	Clusters are **disjoint and exhaustive** (no overlap, no omission).

Table 5: K-Means summary

❷ How Many Clusters? (choosing K)

One of the main limitations of K-Means is that the number of clusters K must be **specified in advance**. However, in real-world data, the true number of natural groupings is **usually unknown**. Choosing the wrong K can lead to:

- **Underfitting**: too few clusters $\Rightarrow$ merging different groups

- **Overfitting** too many clusters $\Rightarrow$ splitting coherent groups

The common methods to choose K are:

- **Elbow Method** looks at the **total within-cluster variation** (also called inertia or distortion), defined as:

$$W(K) = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_K} \|\mathbf{x}_i - \mu_k\|^2 \tag{40}$$

  - $W(K)$ is the total within-cluster variability (it is explained in depth on page 50). It measures **how compact the clusters are**, the smaller $W(K)$, the tighter the points are around their centroids.

  - When $K = 1$, all points are in a single cluster, so $W(1)$ is very high. As K increases, **clusters** become **smaller and tighter**, then $W(K)$ **decreases**. In the extreme case $K = n$, each point is its own cluster, then $W(n) = 0$.

    So, $W(K)$ **always decreases as K increases**, but <u>not</u> linearly.

We compute Elbow Method for different values of K, and plot $W(K)$ vs K. The reason we plot the **within-cluster sum of squares $W(K)$** against the **number of clusters K** is to **visualize how the clustering quality improves** as we increase the number of clusters.

Plotting $W(K)$ against K helps us **identify the point where adding more clusters gives diminishing returns**. It is called the "*elbow method*" because of the shape of the plot. When plotting $W(K)$ versus K, the curve usually drops sharply at first because adding more clusters

quickly improves clustering. After a certain point, however, adding more clusters only yields small improvements. The resulting plot often resembles a bent arm. The "*elbow*" is the point of maximum curvature, where the curve begins to flatten out. This point is important because **before the elbow**, we see **significant gains in reducing** within-cluster variance. **After the elbow**, however, the **gains are minimal**, indicating that increasing model complexity is not worthwhile.

Figure 8: This is an example of the *elbow method* applied to K-means clustering. In the plot, we can see how the within-cluster sum of squares ($W(K)$) changes as the number of clusters K increases from 1 to 10.

- When $K = 1$: $W(K)$ is very large (around 2500), since all points belong to a single, poorly-fitting cluster.

- From $K = 1$ to $K = 3$: $W(K)$ decreases rapidly, showing tat adding clusters significantly improves the clustering.

- From $K = 4$ onward: the curve starts to flatten, meaning adding more clusters gives only small improvements.

The elbow is visible at $K = 4$. This is where the rate slows down sharply. Adding more clusters doesn't significantly reduce $W(K)$. It balances clustering quality with model simplicity.

- **Silhouette Score** (or **Silhouette Coefficient**) evaluates **how well each point fits within its cluster** versus others. The Silhouette Coefficient for a point is:

$$s = \frac{b - a}{\max(a, b)} \tag{41}$$

Where:

- a is the average distance to points in the same cluster

  – b is the average distance to points in the nearest other cluster

  Values close to 1 mean well-clustered points. We compute the average Silhouette score for all points and pick the K that maximizes it.

- **Gap Statistics** (too advanced) compares the total within-cluster variation for the real data to that for randomly generated data with no structure. A large "gap" means the real data forms more distinct clusters. Choose the smallest K for which the gap is maximal or within 1 standard deviation of the maximum.

Example 2: K-Means

Below is a simple run of the K-means algorithm on a random dataset.

- Iteration 0 - **Initialization**
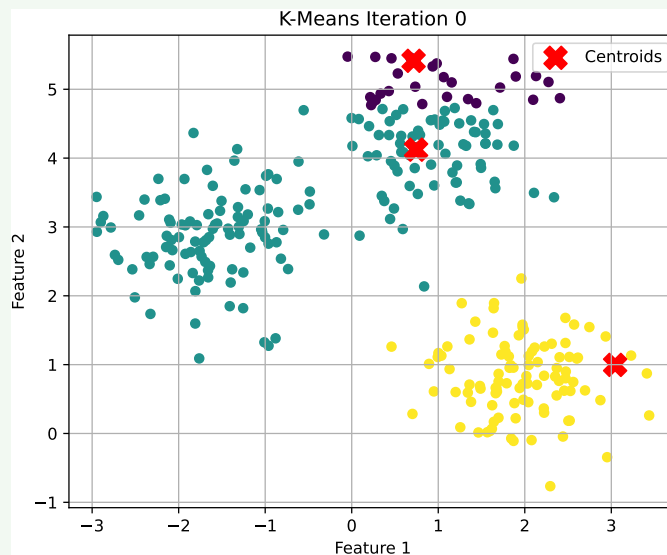
  This is the starting point of the K-Means algorithm. **Three centroids are randomly placed in the feature space**. At this point, no data points are assigned to clusters yet, or all are assumed to be uncolored/unclustered. The positions of the centroids will strongly influence how the algorithm proceeds.

  The goal here is to start with some guesses. The next step will use these centroids to form the initial clusters.

- Iteration 1 - **First Assignment and Update**

46

Each data point is assigned to the closest centroid, forming the first version of the clusters. New centroids are computed by taking the average of the points in each cluster. We can already see structure forming in the data, as points begin grouping around centroids.

This step is the first real clustering, and centroids begin to move toward dense regions of data.

- Iteration 2 - **Re-Assignment and Refinement**

Clusters are recomputed based on updated centroids. Many points remain in the same clusters, but some may shift to a new cluster if a centroid has moved. Centroids continue moving closer to the center of their respective groups.

The algorithm is now refining the clusters and reducing the total distance from points to centroids.

- Iteration 3 - **Further Convergence**

At iteration 3, the K-Means algorithm reached convergence. The centroids no longer moved, and no points changed cluster.This means:

- The algorithm has found a locally optimal solution.

- Further iterations would not improve or change the clustering.

- The final configuration is considered the result of the algorithm.

In practice, this is how K-Means stops: it checks whether the centroids remain unchanged, and if so, it terminates automatically.

2.5.3   Initialization Issues in K-Means

❷ Why initialization matters

The K-Means algorithm starts by choosing K **initial centroids**. These **starting positions are crucial** because K-Means builds the entire clustering process based on them.

Since K-Means is an **iterative algorithm**, it gradually improves the clusters from where it begins. However, if it starts from a bad configuration (i.e. bad initial centroids), it can easily get stuck in a **local minimum**, a clustering solution that isn't optimal but looks good from that starting point.

This means that:

- **Different initial centroids** can lead to **different final results**.

- Some clusters ma be poorly formed or even empty.

- The quality of the final clustering can vary, even on the same data.

⚠ What can go wrong

- **Centroids too close**: if initial centroids are placed near each other, they might all end up capturing the same group of points, leaving other areas underrepresented.

- **Unbalanced clusters**: a poor start can lead to clusters with very different sizes or shapes, even if the data has nicely separated groups.

- **Empty clusters**: it's possible that some centroids end up with no points assigned, which breaks the algorithm's assumptions.

✅ How to improve initialization (conceptually)

To avoid these problems, two common strategies are used:

1. **Multiple random initializations**. Instead of relying on just one random start, K-Means is **run several times with different initial centroids**. The **result with the lowest total distance is selected**. This increases the change of finding a good clustering.

2. **Smart initialization**. Instead of choosing all centroids randomly, a better approach is to choose the first centroid randomly, then pick the next one in a way that they are **as far apart as possible**. This increases the chances of covering different regions of data from the beginning.

   This smarter approach significantly reduces the risk of bad clustering and makes the algorithm more stable and reliable.

2.5.4   K-Means as an Optimization Problem

Given:

- A set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ of n data points in $\mathbb{R}^d$

- A chosen number of clusters K

The goal of K-Means is to **partition the data into K clusters**:

$$C = \{C_1, C_2, \ldots, C_K\}$$

So that a measure of **within-cluster variability** is **minimized**.

❷ What is Within-Cluster Variability?

In simple terms:

- **Cluster**: a group of points that are similar (close together).

- **Variability**: how spread out the points are.

- **Within-Cluster Variability** how spread out the points are **inside each cluster**.

So within-cluster variability measures **how close the points in the same cluster are to each other**. We have low variability when the points are tightly packed and high variability when the points are spread out.

❷ Okay, but why do we need to minimize within-cluster variability?

Because a **good cluster** should contain **points that are close together**, not randomly scattered. So K-Means algorithm tries to form clusters where the **internal "spread" is as small as possible**.

❷ Objective Function: what are we minimizing?

For each cluster C_k, define its **centroid $\mathbf{c}_k$** as the **mean of the points in the cluster**. The within-cluster variability is measured by the following:

$$W\left(C_k\right) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{d} \left(x_{ij} - x_{i'j}\right)^2 \tag{42}$$

- Each observation is a point:

$$\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id}) \in \mathbb{R}^d$$

That means $\mathbf{x}_i$ is a vector with d features.

- $\displaystyle\sum_{i,i' \in C_k}$ means we are looking at **all pairs of points** i and i' inside the same cluster C_k.

- $\sum_{j=1}^{d} (x_{ij} - x_{i'j})^2$, for each feature j, we compute the **squared distance** between point i and point i'.

- $\sum_{i,i' \in C_k} \sum_{j=1}^{d} (x_{ij} - x_{i'j})^2$ measures the **total squared distance** between **all point pairs inside the cluster**.

- $\frac{1}{|C_k|}$, we divide by the number of points to get an **average distance**.

So $W(C_k)$ is the **average squared distance between all pairs of points inside cluster** C_k.

⏱ Optimization Criterion

The full K-Means objective is to **find the clustering** that **minimizes the total within-cluster variability**:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k) \tag{43}$$

This means we are searching for the best possible partition $\{C_1, \ldots, C_K\}$ that results in clusters where **points are as close as possible to each other**, i.e., the clusters are compact.

In other words, we want **every cluster to be compact**. So we add up the within-cluster variability for all clusters. We minimize the problem while keeping it as compact as possible.

✅ Interpretation

A **good clustering** has **low within-cluster variability**: points in the same cluster are close together. Also, K-Means solves an optimization problem where it tries to find such a clustering by minimizing a specific cost function.

2.5.5   Algorithm

The K-Means algorithm is an iterative method that alternates between assigning data points to clusters and updating the cluster centroids, with the goal of minimizing within-cluster variability.

0. **Choose the number of clusters K**. Before running the algorithm, we must decide the number of clusters K. Each cluster will have:

   - A **set of points** $C_1, \ldots, C_K$
   - A **centroid** $\mathbf{v}_1, \ldots, \mathbf{v}_K$
   - A **partition matrix** $\Gamma = [\gamma_{ij}]$ where:

   $$\gamma_{ij} = \begin{cases} 1 & \text{if } \mathbf{x}_j \in C_i \\ 0 & \textbf{otherwise} \end{cases}$$

1. **Initialization**. Randomly assign each observation to one of the K clusters. Then compute the **initial centroids** for each cluster:

   $$\mathbf{v}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

   These are the **mean vectors** of the assigned points.

2. **Assignment**. For each point $\mathbf{x}_i$, compute the distance to all centroids, and assign it to the cluster with the **closest centroid**. This updates the **partition matrix** Γ.

3. **Update Centroids**. After the new assignments, **recompute the centroid** of each cluster using the updated points:

   $$\mathbf{v}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i$$

Repeat Step 2 and Step 3 until:

- ✔ No points change cluster (convergence), or
- ✔ A stopping criterion is met (e.g., max iterations)

📗 **Properties of the algorithm**

✔ **Good**

- – The algorithm **always converges** (it stabilizes after a finite number of iterations)
- – Simple and fast

✘ **Bad**

- – It does **not always find the global optimum**
- – The **results depends** heavily on the **initialization**

In practice, K-means clustering is **typically run multiple times with different random initializations**, and the **result with the lowest cost is kept**.

2.5.6 Limitations

While K-Means is simple and efficient, it has several important limitations:

1. **We must choose the number of clusters K in advance**. Unlike **hierarchical clustering**, which builds a full tree of clusterings, K-Means requires we to **fix K before the algorithm starts**.

   ✖ Choosing the wrong K can lead to poor results.

   ✖ There's no universal rule for selecting K; methods like the Elbow or Silhouette must be used.

2. **Assumes clusters are globular (spherical) in shape**. K-Means tends to form **round-shaped clusters** due to the use of **Euclidean distance**.

   ✖ It **struggles with elongated, irregular, or non-convex shapes**.

   ✖ Works best when clusters are **well-separated, equally sized, and dense**.

3. **Fails with clusters of different sizes or densities**. If one cluster is very dense and another is sparse, K-Means might:

   ✖ Split one true cluster into several pieces

   ✖ Merge multiple small clusters into one

   This happens because Euclidean distance doesn't account for cluster **spread or shape**.

4. **Sensitive to outliers**. K-Means uses **centroids (means)** to define clusters. Since the **mean is sensitive to extreme values**, an outlier can significantly shift a centroid and distort the clustering.

   As an alternative, we could use **K-Medoids**, which is **based on medians rather than means** and is more robust to outliers.

5. **Sensitive to feature scaling and representation**. Because K-Means relies on distance, the algorithm is:

   ✖ Affected by **unscaled or poorly represented data**

   ✖ Sometimes improved by **dimensionality reduction** (e.g., PCA)

2.6   Gaussian Mixture Models (GMMs)

2.6.1   Introduction

A **Gaussian Mixture Model (GMM)** is a **probabilistic model** that assumes the data is generated from a **mixture of several Gaussian distributions** (also called normal distributions), each representing a different **latent group or cluster**.

Each data point is not assigned to one specific cluster (as we have seen in the K-means algorithm), but is instead modeled as **having a probability of belonging to each cluster**.

❷ Why is it called a "*mixture*"?

Because the model represents the **overall data distribution** as a **weighted sum of several Gaussian components**. Mathematically, the probability density function is:

$$f(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x \mid \mu_k, \Sigma_k)$$

Where:

- K is the number of **Gaussian components** (clusters)

- π_k is the **mixing coefficient** of component k (like a *weight*), with $\sum \pi_k = 1$

- $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ is the **multivariate Gaussian distribution** with:
    - Mean μ_k
    - Covariance matrix Σ_k

A more detailed explanation can be found on page 61.

Remark: What is the Multivariate Gaussian Distribution?

❷ What is a 1D Gaussian (Normal) Distribution?

The **1D Gaussian** (also called **Normal Distribution**) is the classic bell curve and it tells us how likely we are to get values near the **mean** μ, and how unlikely values far from the mean are. The spread is controlled by the **standard deviation** σ.

In other words, it describes a **continuous** variable whose values are **most likely to occur near a central point**, and **less likely the farther we move away**.

The **Probability Density Function (PDF)** of a 1D Gaussian is:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \qquad (44)$$

- x is the value of the random variable.

- μ is the **mean**, the center of the distribution.

- σ is the **standard deviation**, it controls the "spread" of the curve.

The bell-shaped curve is symmetric around the mean μ. Most values fall within:

- $\mu \pm \sigma$: $\approx 68\%$

- $\mu \pm 2\sigma$: $\approx 95\%$

- $\mu \pm 3\sigma$: $\approx 99.7\%$

For example, if we have a 1D Gaussian with $\mu = 0$, $\sigma = 1$, it is called the **standard normal distribution**, where the curve peaks at 0 and spreads out smoothly in both directions.

❓ What is a 2D Gaussian Distribution?

The **2D Gaussian** (or **Bivariate Normal Distribution**) is the natural extension of the 1D Gaussian to **two dimensions**. It tells us how likely a point (x, y) is to appear in a **2D space**, just like how the 1D Gaussian tells us about single values on a line.

A **2D Gaussian distribution** is completely defined by:

1. **Mean Vector $\boldsymbol{\mu} \in \mathbb{R}^2$**

   This is the **center** (or peak) of the distribution.

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \tag{45}$$

   It tells us the **average position** of the points, the location where the distribution is highest.

2. **Covariance Matrix $\Sigma \in \mathbb{R}^{2 \times 2}$**

   This tells us how the data is **spread out and oriented** in 2D space.

$$\Sigma = \begin{bmatrix} \sigma_x^2 & \rho \sigma_x \sigma_y \\ \rho \sigma_x \sigma_y & \sigma_y^2 \end{bmatrix} \tag{46}$$

   The diagonal values control the **spread** along each axis, and the off-diagonal values (ρ) represent **correlation** between x and y.

   - σ_x^2, σ_y^2 is the variances along the X and Y axes.
   - ρ is the correlation between x and y.
     - If $\rho = 0$: the axes are independent.
     - If $\rho > 0$: the ellipse tilts upward (positive correlation).
     - If $\rho < 0$: the ellipse tilts downward (negative correlation).

In short, a 2D Gaussian is a "bell-shaped hill" in 3D space, centered at $\boldsymbol{\mu}$, with shape and tilt determined by Σ.

Figure 9: 1D Gaussian and 2D Gaussian (as Contours). In the **1D plot** (left), we see the classic **bell-shaped curve**. This shows the probability density of a variable x under a Gaussian distribution centered at $\mu = 0$ with standard deviation $\sigma = 1$.

- The peak occurs at the mean.

- The density decreases symmetrically as we move away from the mean.

- Most of the probability mass lies within ± 1 or ± 2 standard deviations.

This curve describes how **likely** different values of x are under the distribution. The second plot (right) shows a **2D Gaussian distribution using contour lines**. This represents a Gaussian defined over two variables, x and y.

- The **concentric ellipses** indicate regions of equal probability density (like topographic maps).

- The ellipses are **tiled**, showing that the variables x and y are **positively correlated**.

- The closer to the center, the higher the density.

- A 2D Gaussian distribution models probabilities over two variables $\mathbf{x} = (x, y)$. At each point (x, y), the distribution gives a **density value**, that is, how likely it is that a data point would occur at that location. This means the 2D Gaussian is actually a function from 2D to 1D:

$$f(x, y) = \text{density}$$

  So even though it's defined in 2D, it **outputs a single number** (the density).

This view helps visualize the **shape and orientation** of the distribution in 2D space.

Figure 10: A 3D plot showing the same 2D Gaussian as a surface in three dimensions.

- The horizontal plane is the x-y space.

- The vertical axis shows the **density** at each point.

- The result is a **bell-shaped hill**, centered at the mean.

- The ridge direction and slope reflect the **correlation and spread** between the two variables.

We use x and y as the horizontal plane (input space), and z-axis as **density value** $f(x, y)$. Now, we are plotting the **full shape of the probability surface**: a bell-shaped hill where the height indicates the likelihood of a given point. We add a **third axis to represent the function output**, the density. While **2D contours** are useful for seeing **direction**, **correlation**, and **structure**, a **3D surface** provides a **better sense of the bell shape**.

✅ What is the Multivariate Gaussian Distribution?

The **Multivariate Gaussian Distribution** is simply a **generalization of the 1D and 2D Gaussian** to **any number of dimensions**. If we have a random vector $\mathbf{x} \in \mathbb{R}^d$, then: a **multivariate Gaussian** models the probability of observing different values of that vector, assuming that all components jointly follow a Gaussian distribution.

√x **Formal Definition**

A random vector $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ follows a multivariate normal distribution with:

- **Mean vector**: $\boldsymbol{\mu} \in \mathbb{R}^d$

- **Covariance matrix**: $\Sigma \in \mathbb{R}^{d \times d}$

We write this as:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma) \tag{47}$$

Its **probability density function** is:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \cdot \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{48}$$

- $\boldsymbol{\mu}$ is the **center** (expected value) of the cloud of points.

- Σ is the **shape and orientation** of that cloud

  - The diagonal entries σ_{ii} control how spread out the data is in each direction.
  - The off-diagonal entries σ_{ij} represent **correlation** between different variables.

So a multivariate Gaussian forms an **ellipsoidal distribution** in high-dimensional space.

❓ Why GMM uses Multivariate Gaussian Distribution?

In Gaussian Mixture Models, each component is a **multivariate Gaussian**:

- It models a cluster in **multidimensional space**.

- The full GMM is a **weighted sum** of these multivariate Gaussians.

This allows GMM to **model complex, overlapping, and anisotropic (non-spherical) clusters** much better than K-Means.

🗐 Relation to K-Means

K-Means is a **special case of GMM**:

- K-Means assigns each point to **only one cluster**, called **Hard Clustering**.

- GMM assigns each point a **probability of belonging** to each cluster, called **Soft Clustering**.

In fact, **if we assume that**:

- All clusters have **equal, spherical** covariance.

- Each point is assigned to the **closes cluster center**.

Then *GMM reduces to K-Means.*

❷ But why should we use GMM instead of K-means?

Four reasons:

1. **Soft Clustering vs Hard Clustering**

   - **Hard Assignment**: K-Means assigns each point to **only one cluster**.
   - **Soft Assignment**: GMM assign each point a probability of belonging to each cluster.

   GMM is useful when clusters overlap, we're uncertain or want probabilistic interpretation, or we want to model real-world uncertainty (e.g., "60% likely to be in Cluster 1").

2. **Flexible Cluster Shapes**

   K-Means assumes clusters are **spherical and equally sized**. Instead, GMM allows:

   - Elliptical shapes;
   - Different sizes and orientations;
   - Correlations between features.

   This is because GMM uses **covariance matrices**, whereas K-Means just uses Euclidean distance.

3. **Statistical Foundation**

   K-Means is an algorithm that minimizes distance. Instead, GMM is a **probabilistic model** with a well-defined likelihood function. Then, we can estimate the uncertainty and use model selection to determine the number of clusters.

4. **K-Means is a Special Case of GMM**

   If we assume:

   (a) All clusters have **equal and diagonal covariance matrices** (no correlation);
   (b) Use hard assignments.

   Then GMM becomes K-Means. So GMM is a **more general, more powerful model**.

However, GMM is not always the best method to consider. It depends on the data and the final goal:

- Use **K-Means** when **clusters** are clearly **separated**, **round**, and **fast results** are needed.

- Use **GMM** when we want a **flexible**, **probabilistic**, and **realistic model** of how our data is generated.

Feature	K-Means	GMM
Assignment	Hard	Soft (probabilistic)
Cluster shape	Spherical	Elliptical, flexible
Model type	Geometric	Probabilistic (generative)
Speed	Fast	Slower (EM algorithm)
Interpretability	Simple centroids	Full statistical model

Table 6: Trade-offs between K-Means and GMM.

2.6.2   Mathematical Foundation

Even though we have already introduced the GMM formula, we provide a thorough explanation here.

❷ What is a Mixture Model?

A **mixture model** assumes that our data was generated from **multiple probability distributions**, not just one. For GMMs, we assume: each data point $\mathbf{x}_i \in \mathbb{R}^d$ was **generated by one of several multivariate Gaussian distributions**. But we don't know:

- *Which point came from which component.*

- *What the component parameters are.*

▤ GMM Definition

The probability density function of a Gaussian Mixture Model with K components is:

$$p\left(\mathbf{x}\right) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}\left(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k\right) \tag{49}$$

Where:

- $\mathbf{x} \in \mathbb{R}^d$ is the observed data point.

- π_k is the **mixing coefficient** for component k, with $\sum_{k=1}^{K} \pi_k = 1$, and $\pi_k \geq 0$.

- $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$ is the **multivariate Gaussian density** of the k-th component, with:

    - Mean $\boldsymbol{\mu}_k \in \mathbb{R}^d$
    - Covariance matrix $\Sigma_k \in \mathbb{R}^{d \times d}$

  This is the **probability density** of the Multivariate Gaussian Distribution, also called the **likelihood of observing the point x**, **given the parameters of component** k (mean μ and covariance Σ).

  For example, imagine we're standing at a point $\mathbf{x}$ in space, and asking: *how likely is it that a point like this came from Gaussian cluster k?* The answer is:

  $$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$$

  The value is **higher** if $\mathbf{x}$ is **closer to the center** of cluster k, and **lower** if it's farther away.

The meaning of each term is as follows:

Symbol	Meaning
π_k	Prior probability of choosing cluster k.
$\boldsymbol{\mu}_k$	Mean (center) of the k-th Gaussian component.
Σ_k	Covariance (shape and orientation) of the k-th component.
$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \Sigma_k)$	Likelihood of $\mathbf{x}$ under the k-th Gaussian.
$p(\mathbf{x})$	Overall probability of observing $\mathbf{x}$ under the full model.

To generate a data point $\mathbf{x}$, first choose a component k with probability π_k, then draw $\mathbf{x}$ from the Gaussian $\mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$. This is why GMM is a generative model, **it models how data could have been generated**.

2.6.3   Responsibilities

Unlike **K-Means**, where each point is assigned to exactly **one cluster** (hard clustering), **GMM** performs **soft clustering**: each point has a probability of belonging to every cluster. These probabilities reflect *how likely* it is that a point came from each component of the mixture.

❷ Why do we need to introduce Responsibilities in GMM?

⚠ We have a **problem** at the moment. When using a Gaussian Mixture Model, we assume that each data point $\mathbf{x}_i$ was generated by **one of the** K **Gaussian components**. But, we **don't know which one**. That information is **hidden** or **latent**. We only observe:

- The data point $\mathbf{x}_i$

- The parameters of the Gaussian $(\pi_k, \boldsymbol{\mu}_k, \Sigma_k)$

We don't observe which component k generated $\mathbf{x}_i$.

I still don't understand what the problem is. In a Gaussian Mixture Model, we assume that each data point $\mathbf{x}_i$ is generated by **one of the** K **Gaussian components**. So the true process that generates each point is:

1. Randomly choose a component k with probability π_k.

2. Generate a point $\mathbf{x}_i$ from the Gaussian $\mathcal{N}(\mu_k, \Sigma_k)$

But we don't know which Gaussian generated each point and this is the core issue:

- We **observe only the data points** $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

- We **do not observe the "label"** of which component (cluster) generated each point.

That **label is missing**! This is problematic because, if we want to **train a model** (i.e., estimate the π_k, μ_k, and Σ_k), we would **ideally want to group the data by component**. However, we can't do that because we don't know which point belongs to which component.

Okay, but if we're using the GMM formula, then we know everything about each cluster that produces each point. That's a good point, but the practice is slightly different. In the real world, especially in machine learning, we don't know the origin of each point because we only have the observed data. So, we are trying to **infer** the underlying **clusters**, **without knowing** which one generated each point.

> **Example 3: analogy of the GMM problem**
>
> Imagine a factory that has 3 machines (A, B, C) making cookies. Each machine:
>
> - Adds a different amount of sugar and chocolate

- Sometimes makes similar-looking cookies, but not exactly the same

- Has a different production rate (e.g., Machine A makes 50% of all cookies, B makes 30%, C makes 20%)

Now, someone gives us a box of mixed cookies and says: "*These were made using machines A, B, and C, but I'm not telling you which cookie came from which machine.*" We taste each cookie and write down how sweet and chocolatey it is. That's our data.

⚠ **The Problem.** We don't know:

- Which machine made each cookie.

- How sweet/chocolatey each machine's recipe really is.

- How often each machine is used.

In GMM terms:

- We have the data $\mathbf{x}_1, \ldots, \mathbf{x}_n$ (cookie features).

- But the label "*this came from machine A*", is hidden.

✅ **The Solution.** Since we can't know for sure where each cookie came from, we say: "*Let me estimate how likely it is that this cookie came from each machine.*". So for Cookie #1, we might say:

- 70% Machine A

- 20% Machine B

- 10% Machine C

This set of numbers is the responsibility vector for Cookie #1. We do this for every cookie. Now, even though we don't know the true labels, we have soft guesses, and that's enough to:

- Estimate what each machine's recipe looks like (mean and variance).

- Estimate how many cookies each machine probably made (mixing proportions).

The core idea is that we need "responsibilities" because the cluster label is missing. Rather than guessing the labels directly, we estimate the probability that each point belongs to each cluster. These probabilities are called "responsibilities".

✅ Solution: Responsibilities = Posterior Probabilities

We come back to GMM. We assume that:

1. Each point $\mathbf{x}_i$ was generated by **one of K Gaussians**.

2. But we **don't observe** which one.

So we introduce a **latent variable $z_i \in \{1, \ldots, K\}$** that **represents the unknown cluster** of $\mathbf{x}_i$ (hidden cluster assignments).

We introduce the idea of a **latent variable z_i**, which says: "let's pretend each point $\mathbf{x}_i$ has a hidden label $z_i \in \{1, \ldots, K\}$". But since we don't observe z_i, we'll estimate the **probability** that it takes each value.

A **Latent Variable** is a variable that:

1. **Exists conceptually**, and

2. **Affects the data**, but

3. Is **not directly observed** in our dataset

In other words, a latent variable is something we don't see, but it explains patterns in the things we do see.

> **Example 4: continuation of the example on page 63**
>
> Let's go back to our cookie story.
>
> - We **see the cookie** $\Rightarrow$ that's observed data.
>
> - We **don't see the machine** that made it $\Rightarrow$ that's a **latent variable**.
>
> The **machine label** (e.g. A, B, or C) is a latent variable z because:
>
> 1. It determines the cookie's sweetness/chocolate level.
>
> 2. But we aren't told what it is, we must infer it.

Since we cannot observe z_i directly, we can **estimate the probability** that $z_i = k$ for each component k. This estimate is called **Responsibility** and is mathematically expressed as follows:

$$\gamma_{ik} = P\left(z_i = k \mid \mathbf{x}_i\right) \tag{50}$$

We are trying to *approximate* the unknown values of z_i. This enables us to **learn from incomplete data** without needing the true labels.

Latent Variable	Responsibility
Missing truth	Out best guess of that truth.

The responsibility is computed using **Bayes' Rule**:

$$\gamma_{ik} = \frac{\pi_k \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k\right)}{\displaystyle\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j\right)} \tag{51}$$

In simple terms, we have:

- The numerator is: how well component k explains point $\mathbf{x}_i$, weighted by how likely that component is in general.

- The denominator ensures that the total probability sums to 1 (normalization across all components).

The **main idea** behind GMM is very simple. Rather than stating that "point i is in cluster 2", GMM says that point i is 80% likely to be in cluster 2, 15% likely to be in cluster 1, and 5% likely to be in cluster 3. These values are the **responsibilities** γ_{ik}. The word *responsibility* reflects that **each component is partially responsible** for explaining each point.

2.6.4 Expectation-Maximization (EM) Algorithm

The **Expectation-Maximization (EM) Algorithm** is how we **train** a Gaussian Mixture Model (GMM). Meaning, how we **estimate** the unknown parameters:

- Mixing coefficients π_k

- Means $\boldsymbol{\mu}_k$

- Covariances Σ_k

We use the EM algorithm **due to the unknown cluster labels**, z_i. The data is incomplete, so we cannot group the points by component. The EM algorithm usually works when some variables are latent (hidden).

✖ How EM Works (Big Picture)

EM is an **iterative algorithm** with two main steps:

1. **E-step (Expectation)**. In this step, we **compute responsibilities**:

$$\gamma_{ik} = P\left(z_i = k \mid \mathbf{x}_i\right) = \frac{\pi_k \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_k, \Sigma_k\right)}{\displaystyle\sum_{j=1}^{K} \pi_j \cdot \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_j, \Sigma_j\right)}$$

   This is the **expected value of the latent variable** z_i, given the current parameter estimates. In simpler terms: we update our guess of how much each cluster k is responsible for each point $\mathbf{x}_i$.

2. **M-step (Maximization)**. In this step, we **update the model parameters** using the responsibilities:

   (a) **Update weights** π_k:

$$\pi_k = \frac{1}{n} \cdot \sum_{i=1}^{n} \gamma_{ik}$$

   (b) **Update means** $\boldsymbol{\mu}_k$:

$$\boldsymbol{\mu}_k = \frac{\displaystyle\sum_{i=1}^{n} \gamma_{ik} \cdot \mathbf{x}_i}{\displaystyle\sum_{i=1}^{n} \gamma_{ik}}$$

   (c) **Update covariances** Σ_k:

$$\Sigma_k = \frac{\displaystyle\sum_{i=1}^{n} \gamma_{ik} \cdot \left(\mathbf{x}_i - \boldsymbol{\mu}_k\right) \cdot \left(\mathbf{x}_i - \boldsymbol{\mu}_k\right)^T}{\displaystyle\sum_{i=1}^{n} \gamma_{ik}}$$

We're **maximizing the likelihood** of the data under the current soft assignments.

We alternate E and M steps **until the log-likelihood stops improving significantly**, or we **reach a maximum number of iterations**. This gives us the best estimate of the parameters, despite not knowing the cluster labels.

❓ What EM is really doing

Each iteration:

- Uses the **E-step** to "*guess*" hidden information (soft labels).

- Uses the **M-step** to "*retrain*" the model based on those guesses.

That's why it's called **Expectation-Maximization**:

- We **expect** the missing values.

- Then **maximize** the likelihood given that expectation.

2.6.5   Comparison between GMM and K-Means

Even though both Gaussian Mixture Models (GMM) and K-Means are used for **clustering**, they differ significantly in their assumptions, outputs, and flexibility.

- **Conceptual Difference**

Concept	K-Means	GMM
Type	Geometric algorithm	Probabilistic generative model
Goal	Minimize distances to centroids	Maximize likelihood of data under Gaussian model
Output	Hard assignments (1 cluster)	Soft assignments (probabilities per cluster)

- **Shape and Structure**

Feature	K-Means	GMM
Cluster Shape	Spherical (equal radius)	Elliptical (any orientation/size via covariance)
Covariance Info	Not modeled	Fully modeled via Σ_k for each cluster
Equal size	Assumes clusters are similar in size	Allows different size and shape per cluster

- **Assignment Method**

  - **K-Means** assigns each point to the **nearest centroid** (based on Euclidean distance).

  - **GMM** assigns each point a **probability** of belonging to each cluster (based on likelihood).

  This makes GMM better for:

  - Overlapping clusters.

  - Uncertain or fuzzy boundaries.

  - Data with correlated features.

- **Optimization Strategy**

  - **K-Means** minimizes **within-cluster sum of squared distances**.

  - **GMM** maximizes the **log-likelihood** of the observed data via the **EM algorithm**.

- **Computational Cost**

Aspect	K-Means	GMM
Speed	Faster	Slower (EM is iterative)
Convergence	Guaranteed	Also guaranteed, but may find local optima
Initialization	Important	Even more sensitive to initialization

- **Summary Table**

Feature	K-Means	GMM
Assignments	Hard	Soft (probabilities)
Cluster shape	Spherical	Elliptical (via covariance)
Uses probability model?	✖ No	✔ Yes
Optimization objective	Distance	Log-likelihood
Suitable for overlapping?	✖ No	✔ Yes

- **When to Use What?**

Use Case	Choose
We need fast, simple partitioning	K-Means
We want a statistical model or density info	GMM
Clusters overlap or vary in shape	GMM
We have very large datasets	K-Means (or approximate GMM)

3 Discriminant Analysis

3.1 From Unsupervised to Supervised

So far, we've been working with **GMMs** and **clustering**, which are **unsupervised** methods. This meaning:

- We had no labels.

- We tried to discover patterns (e.g., clusters) **just from the data**.

Now we're moving to **Supervised Learning** where:

- Each observation $\mathbf{x}_i$ **comes with a label** $y_i \in \{1, \dots, K\}$

- We **want to learn a function** that can classify new inputs based on past labeled data.

❷ Why This Transition Is Natural (GMM → LDA)

Let's revisit how we modeled things in GMM. In GMM, we assumed:

$$\mathbf{x} \sim \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(\mu_k, \Sigma_k)$$

But we **didn't know** which k each $\mathbf{x}$ came from, hence, EM algorithm. In Supervised Learning (like Linear Discriminant Analysis, LDA), we **do know** which class k each $\mathbf{x}$ comes from. This lets us directly estimate the parameters π_k, μ_k, Σ_k from the **labeled data**. Thus:

GMM (Unsupervised)	LDA/QDA (Supervised)
No labels	Labels $y_i \in \{1, \dots, K\}$
Estimate hidden assignments (EM)	Directly estimate class parameters
Responsibilities: soft membership	Class probabilities from Bayes rule
Soft clustering	Hard classification

Term	Meaning
Latent Variable	In GMM, the unknown class membership Z; in supervised learning, it's known.
Generative Model	A model for how data is generated: first sample label $y \sim \pi_k$, then $\mathbf{x} \sim \mathcal{N}(\mu_k, \Sigma_k)$.
Supervised Learning	Learn a classifier from labeled data — i.e., a function $f(\mathbf{x}) = \hat{y}$.
Discriminant Analysis	Use Bayes' rule with class-conditional distributions to assign classes.

✖ **Key Formulation Shift**

In GMM (unsupervised), we care about:

$$\mathbb{P}(\text{class } = k \mid \mathbf{x}) = \frac{\pi_k \cdot \phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)}{\displaystyle\sum_{j=1}^{K} \pi_j \cdot \phi\left(\mathbf{x} \mid \mu_j, \Sigma_j\right)}$$

Now in **supervised learning**, we do the same, but:

- We estimate μ_k, Σ_k **per class** from labeled data.

- Then use Bayes' rule to classify a new point $\mathbf{x}$.

This is the basis for LDA and QDA (Quadratic Discriminant Analysis).

We're not throwing away the GMM mindset, we're now using it **with supervision**:

- Same assumptions: Gaussian class-conditional densities.

- But now, we **know the true class**, so we don't need EM.

This is why **LDA** is sometimes called the **supervised version of GMM with equal covariances**.

3.2 Introduction to Supervised Learning

Supervised Learning is about learning a mapping from inputs to outputs using **labeled data**. We're given a dataset:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\} \tag{52}$$

Where:

- $\mathbf{x}_i \in \mathbb{R}^p$: vector of features (e.g., age, salary, pixel values, etc.).

- $y_i \in \{1, 2, \dots, K\}$: label.

Our goal is to **learn a function**:

$$f(\mathbf{x}) = \hat{y} \tag{53}$$

That can predict the correct y for a new input $\mathbf{x}$.

≣ Types of Supervised Learning

Problem Type	Output y	Examples	Algorithms
Classification	Categorical/Discrete	Tumor type	LDA, QDA, logistic reg
Regression	Real-valued	Temperature	Linear regression

⚖ Supervised vs. Unsupervised

	Unsupervised	Supervised
Input	$\mathbf{x}_i$	$(\mathbf{x}_i, y_i)$
Goal	Discover structure (clusters)	Predict outcome y from $\mathbf{x}$
Methods	GMM, K-Means	LDA, QDA

So in GMM, we tried to **infer the labels**. In Supervised Learning, the labels are **given**, so the task becomes more about **learning rules**.

❷ What is Discriminant Analysis, and how is it linked to Supervised Learning?

Because **Discriminant Analysis** is one of the **oldest and most classic forms of supervised learning**, specifically for **classification** tasks.

In Supervised Learning, we want to **learn a rule that predicts the class y given an input $\mathbf{x}$**. Discriminant Analysis **does exactly that**, by:

1. Using the labeled data $(\mathbf{x}_i, y_i)$

2. Estimating how each class looks in feature space, with a **probability model** for $\mathbb{P}(\mathbf{x} \mid y = k)$

3. Applying **Bayes' rule** to invert that: estimate $\mathbb{P}(y = k \mid \mathbf{x})$

4. Choosing the class with the highest probability

So we use the label y_i to:

- Group the data by class;

- Estimate a distribution for each class;

- Then **discriminate** new points by which class distribution they're most likely to belong to.

🔴 **More in-depth. Discriminant Analysis** is a method to **classify observations into categories** (like "spam" vs "not spam", or species A vs species B) by:

1. **Modeling how each class generates data**, using a probability distribution (usually multivariate normal).

2. Then using **Bayes' Rule** to compute the probability that a new point belong to each class.

3. Finally, assigning the new point to the **most probable class**.

The word *discriminant* refers to the idea of **finding rules or boundaries that discriminate** between classes based on the features $\mathbf{x}$. So:

- LDA: uses linear boundaries

- QDA: uses quadratic boundaries

These are both types of discriminant functions.

Discriminant Analysis **is called generative because it is a generative model** that assumes we can model:

- $\mathbb{P}(\mathbf{x} \mid y = k)$: the distribution of features **given** the class.

- $\mathbb{P}(y = k)$: the prior probability of each class.

Using Bayes' theorem:

$$\mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x} \mid y = k) \cdot \mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

This is discriminant analysis, we **use these class-conditional densities to make predictions**.

Definition 1: Generative Model

A **Generative Model** models **how the data is generated**, i.e., it learns the **joint distribution**:

$$\mathbb{P}(\mathbf{x}, y)$$

From which we can **generate new data**, or compute the conditional:

$$\mathbb{P}(y \mid \mathbf{x})$$

More specifically, in classification tasks, a generative model models:

$$\mathbb{P}(\mathbf{x} \mid y = k) \quad \text{(class-conditional density)}$$

And:

$$\mathbb{P}(y = k) \quad \text{(class prior)}$$

Then it uses **Bayes' Rule** to compute:

$$\mathbb{P}(y = k \mid \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x} \mid y = k) \cdot \mathbb{P}(y = k)}{\mathbb{P}(\mathbf{x})}$$

Example 1: LDA as a Generative Model

LDA assumes:

- Each class k has a Gaussian distribution:

$$\mathbf{x} \mid y = k \sim \mathcal{N}(\mu_k, \Sigma)$$

- We learn:
    - μ_k, the mean of class k
    - Σ, shared covariance matrix
    - π_k, prior probability of class k

With those, we can:

- **Generate data**: sample a class $y \sim \pi_k$, then sample $\mathbf{x} \sim \mathcal{N}(\mu_k, \Sigma)$

- **Classify new points** using **Bayes' Rule**

✎ Practical Importance

- **Data**: we often have real-world datasets where class labels are known (e.g., in medicine, finance, spam detection).

- **Goal**: we want to *classify* new observations as accurately as possible.

- **Approach**: fit models like LDA, logistic regression, or modern ML classifiers (SVM, Trees, etc.).

3.3   Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a **supervised classification method** that models the probability distribution of each class as a multivariate **Gaussian with the same covariance matrix**, and uses **Bayes' rule** to assign a new observation to the class with the **highest posterior probability**.

Because it assumes equal covariances across classes, the resulting **decision boundaries are linear**, hence the name *Linear* Discriminant Analysis.

❷ What LDA Tries to Do

We're given data from multiple classes (say $K = 2$ or more), and we want to **assign a new point x to the most likely class**. LDA does this by:

1. Modelling how each class generates data (generative model).

2. Using Bayes' rule to compute $\mathbb{P}\left(y = k \mid \mathbf{x}\right)$.

3. Assigning $\mathbf{x}$ to the class with the highest probability.

Example 2: LDA

Suppose we want to classify fruit by weight and color:

- Oranges: heavy, orange.

- Lemons: light, yellow.

LDA will:

- Estimate the **mean** and **spread** (covariance) of orange and lemon data.

- Find a **line** that best separates the two, that's the linear discriminant.

▤ Assumptions Behind LDA

1. **The data in each class looks like a cloud of points shaped like a Gaussian**. Imagine we have two groups of data:

   - Group A, orange points
   - Group B, blue points

   Each group is made of points in 2D (like height and weight). We **assume** that in each group, the points:

   (a) Are **spread out in a bell-shaped way**.
   (b) Cluster around a mean (center).
   (c) Are more dense in the middle, less dense on the sides.

That's called a **multivariate normal distribution** (like a 2D Gaussian blob).

❓ **Why?** Because this makes the math nice. We can use formulas to describe where the data is most likely to appear in each class.

2. **All the groups are spread out in the same way**. This means:

   - Even though Group A and Group B may be **centered in different places** (different means),
   - They are **equally wide, equally tilted, and equally stretched** (same covariance).

Two round clouds of points, one at the left, one at the right, then same shape, just different centers.

❓ **Why?** Because this gives us **linear boundaries**. Meaning, the separator between groups will be a **straight line** (or a plane in higher dimensions). That's why it's called *Linear* Discriminant Analysis.

3. **We roughly know how frequent each class is**. This is called the **prior probability**: if we know that 70% of our training data is oranges and 30% lemons, we should take that into account when classifying.

❓ **Why?** Because we don't want to treat all classes as equally likely if they aren't. Priors help us adjust for that.

📗 Derivation via Bayes Rule

To really understand how LDA works, not just *what it does* but *why it works*, we need to know that LDA classifies a point $\mathbf{x}$ by computing "*which class is most likely to have generated this point?*". And the way to compute that is:

- Estimate how likely $\mathbf{x}$ is *under each class* (the likelihood).

- Multiply it by how common that class is (the prior).

- Then use Bayes' Rule to get the final answer: the **posterior probability**.

✅ **Goal.** Specifically, we want to **assign a new observation**, $\mathbf{x}$, to one of K classes by computing the following:

$$\mathbb{P}(y = k \mid \mathbf{x}) \quad \text{(posterior probability)} \tag{54}$$

It is the **Posterior Probability**, which answers the question, "*given an observation $\mathbf{x}$, what is the probability that it belongs to class k?*" It's called the **posterior** probability because it's what we know **after** seeing the data $\mathbf{x}$. In other words, this is the **probability that x belongs to class** k, given the observed features.

1. **Use Bayes' Rule**. Bayes' Rule helps us to **compute the posterior**:

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right) = \frac{\mathbb{P}\left(\mathbf{x} \mid y = k\right) \cdot \mathbb{P}\left(y = k\right)}{\mathbb{P}(\mathbf{x})}$$

Where:

- $P(y = k \mid \mathbf{x})$ is the posterior, what we want to compute.
- $P(\mathbf{x} \mid y = k)$ is the likelihood, how likely $\mathbf{x}$ is if it comes from class k.
- $P(y = k)$ is the prior, how likely class k is overall.
- $P(\mathbf{x})$ is the evidence, same for all classes, just a normalizing constant.

2. **What LDA Assumes**. LDA **assumes** that:

   (a) Each class k generates data using a **Gaussian distribution**:

   $$\mathbf{x} \mid y = k \sim \mathcal{N}(\mu_k, \Sigma)$$

   (b) All classes share the same covariance Σ.

   (c) The prior $P(y = k) = \pi_k$ is known or estimated from the data.

3. **Simplify the Formula**. Because $\mathbb{P}(\mathbf{x})$ is the same for every class (it doesn't depend on k), we can **ignore it when choosing the best class**. So the posterior is **proportional** to:

   $$\mathbb{P}(y = k \mid \mathbf{x}) \propto \pi_k \cdot \phi(\mathbf{x} \mid \mu_k, \Sigma)$$

   Where:

   - $P(y = k \mid \mathbf{x})$ is the posterior, what we want to compute.
   - $\propto$ is the proportional symbol.
   - $\phi(\mathbf{x} \mid \mu_k, \Sigma)$ is the Gaussian probability density for class k.

   To make this easier to work with, we **take the logarithm** of this expression (log doesn't change which class is biggest). This gives the **Discriminant Function**:

   $$\delta_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \mathbf{x}^T \Sigma^{-1} \mu_k \qquad (55)$$

   Then we classify $\mathbf{x}$ to the class k with the **highest** $\delta_k(\mathbf{x})$.

The final rule is that we need to **assign x to the class k for which $\delta_k(\mathbf{x})$ is the largest**. This is the LDA decision rule.

❷ What is the Decision Rule?

The **LDA Decision Rule** assigns a new observation $\mathbf{x}$ to the class k with the **highest posterior probability**:

$$\hat{y} = \arg\max_k \mathbb{P}(y = k \mid \mathbf{x}) \qquad (56)$$

But since **computing** that posterior **directly is messy**, we used **Bayes' Rule to rewrite it** as:

$$\mathbb{P}(y = k \mid \mathbf{x}) \propto \pi_k \cdot \phi(\mathbf{x}; \mu_k, \Sigma)$$

So instead, we define a **discriminant function**:

$$\delta_k(\mathbf{x}) = \log(\pi_k) - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \mathbf{x}^T \Sigma^{-1} \mu_k$$

Then the decision rule becomes:

$$\hat{y} = \arg \max_k \delta_k \left(\mathbf{x} \right) \tag{57}$$

❷ Geometry: Why It's Called Linear

LDA is called Linear Discriminant Analysis because the decision boundary it creates between classes is a **straight line** (in 2D), or a **hyperplane** (in higher dimensions).

❷ What's a decision boundary? It's the set of points $\mathbf{x}$ where:

$$\delta_k \left(\mathbf{x} \right) = \delta_l \left(\mathbf{x} \right)$$

i.e., where the classifier is **undecided** between class k and class l. This boundary tells us: *"here, both classes are equally likely, it's the tipping point between them."*.

▤ Remark: LDA Discriminant Function. The discriminant function:

$$\delta_k \left(\mathbf{x} \right) = \log \left(\pi_k \right) - \frac{1}{2} \, \mu_k^T \, \Sigma^{-1} \, \mu_k + \mathbf{x}^T \, \Sigma^{-1} \, \mu_k$$
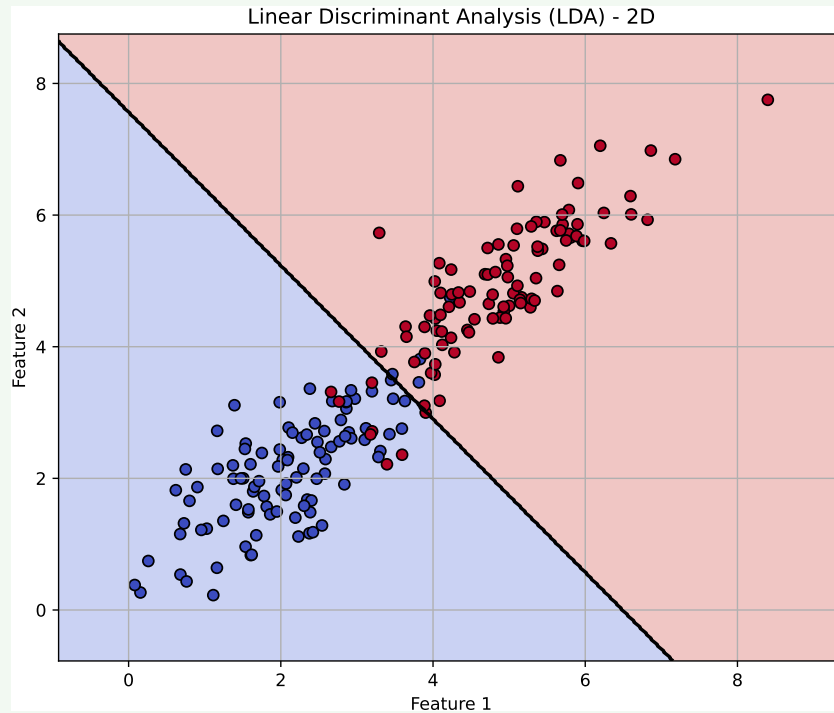
It's **linear in x**, there's no square, no exponent, just a dot product of $\mathbf{x}$ with some vector. So if we compare two classes k and l, their difference:

$$\delta_k \left(\mathbf{x} \right) - \delta_l \left(\mathbf{x} \right)$$

Is also **linear in x**.

Example 3: Plot of a 2D LDA

Let's say we have 2 features: height (cm), weight (kg). Then the LDA boundary between class A and class B will look like a straight line in the 2D plane:

Linear Discriminant Analysis (LDA) - 2D

All the points on the blue side belong to class A, and all the points on the other side belong to class B.

However, LDA support any number of classes $K \geq 2$. With 2 classes, the decision boundary is a single line; with 3 or more classes, LDA creates multiple regions, each with its own boundary between class pairs.

LDA Decision Boundaries (3 Classes in 2D)

LDA Decision Boundaries (4 Classes in 2D)

LDA Decision Hyperplane (3D)
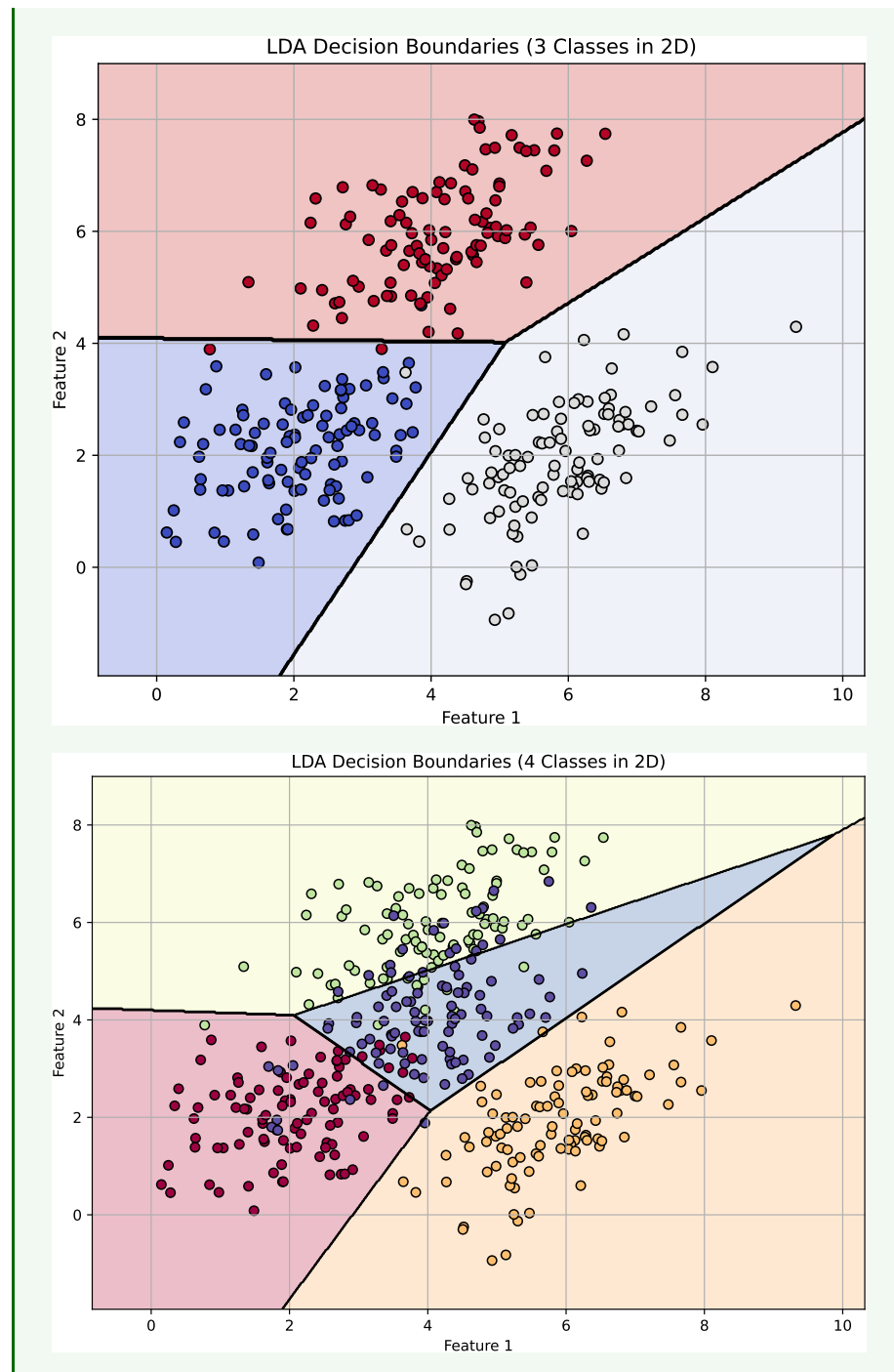
3.4   Quadratic Discriminant Analysis (QDA)

Quadratic Discriminant Analysis (QDA) is a **supervised classification method** that models each class as a multivariate Gaussian distribution **with its own covariance matrix**.

It uses **Bayes' Rule** to classify new observations by computing posterior probabilities, like LDA, but allows **more flexible class shapes**.

🗒 Intuition

QDA is just like LDA, **but more flexible**:

- In **LDA**, all classes share the same shape (same covariance, then same spread, same orientation).

- In **QDA**, each class has its **own shape and orientation** in feature space.

This means we can model curved boundaries and non-symmetric class regions. But the **price is more parameters**, and **more chance of overfitting** on small datasets.

⚠ What is Overfitting?

Overfitting occurs when a **model learns** not just the **true patterns** in the data, **but also the random noise**. It's like memorizing answer for an exam instead of understanding the subject; we do well on the questions we saw before (training data), but fail on new ones (test data).

Imagine fitting a curve to data points:

Fit Type	Description
Underfitting	The model is too simple (e.g., a straight line when the pattern is curved), it misses important structure.
Good Fit	The model captures the true pattern without being too complex.
Overfitting	The model bends and twists to go exactly through every training point, even if those points contain random noise.

Table 7: Fitting type.

Our model is overfitting when we observe the following:

- **Training error**: very *low*

- **Test error**: *high*

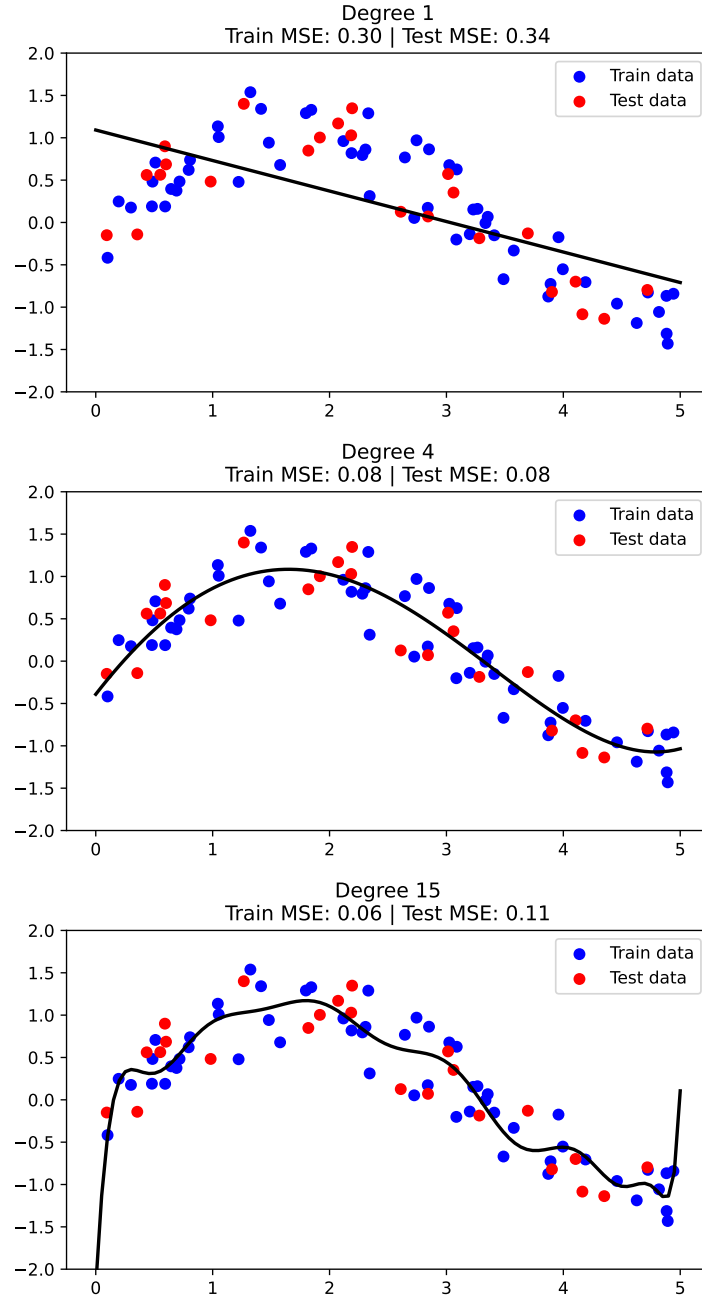The model performs **too well** on the training set because it's **memorized it**, not generalized it.

Figure 11: An example of overfitting. This figure shows polynomial regression models with increasing complexity. Although polynomial regression itself hasn't been explained, the purpose of this figure is to illustrate the effect of overfitting as model complexity increases (degree 1 underfits, too simple, degree 4 fits well, good generalization, degree 15 overfits, captures noise).

⚖️ How does QDA differ from LDA?

- **Covariance Matrix**

LDA	QDA
Shared across all classes, Σ	Unique per class, Σ_k for each class

Covariance Matrix.

- **Decision Boundary**

LDA	QDA
Linear (straight lines / hyperplanes)	**Quadratic** (curved lines / surfaces)

Decision Boundary.

- **Model Complexity**

LDA	QDA
Simpler, fewer parameters to estimate	More complex, more parameters (one full covariance per class)

Model Complexity.

- **Bias vs. Variance**

LDA	QDA
Lower variance, higher bias (more stable)	Lower bias, higher variance (risk of overfitting)

Bias vs. Variance.

- **Shape of Class Regions**

LDA	QDA
All classes modeled with the **same** **shape & orientation**	Each class can have a **different shape & orientation**

Shape of Class Regions.

- **Interpretability**

LDA	QDA
Easier to interpret decision boundaries	Harder to visualize, especially in higher dimensions

Interpretability.

- **Performance with Few Data**

LDA	QDA
Works well, robust	Needs more data to estimate separate covariances reliably

Performance with Few Data.

❷ What's the key difference?

LDA and QDA both model each class as a multivariate Gaussian, but the key difference is:

- **LDA assumes all classes share the same covariance matrix**. The data clouds (ellipses) all have the same shape and orientation.

- **QDA allows each class to have its own covariance matrix**. The data clouds (ellipses) can have different shapes, sizes, and orientations.

❷ What this means in practice

Concept	LDA	QDA
Boundary shape	Straight lines (linear)	Curved lines (quadratic)
Flexibility	Less, all classes "look similar"	More, each class can "look unique"
Math form	Linear discriminant function	Quadratic discriminant function

Figure 12: We have two clouds of points. In LDA, they are both ellipses of the same shape, just located in different places. In QDA, one ellipse might be wide and flat, the other tall and narrow, and the model adapts to those shapes.

❓ The Decision Rule and Why It's Quadratic

We want to classify a new point $\mathbf{x}$ by computing the **posterior probability** (same as page 77):

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right)$$

And assigning $\mathbf{x}$ to the class k with the **highest posterior**. This is the same idea as LDA, but now, each class has **its own covariance matrix**, which changes the math.

1. **Bayes' Rule** (same as page 77).

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right) = \frac{\mathbb{P}\left(\mathbf{x} \mid y = k\right) \cdot \mathbb{P}\left(y = k\right)}{\mathbb{P}\left(\mathbf{x}\right)}$$

   Again, since the denominator $\mathbb{P}\left(\mathbf{x}\right)$ is the same for all classes, we only need to **maximize the numerator**:

$$\mathbb{P}\left(y = k \mid \mathbf{x}\right) \propto \pi_k \cdot \phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)$$

   - π_k is the class prior.
   - $\phi\left(\mathbf{x} \mid \mu_k, \Sigma_k\right)$ is the Gaussian density with **class-specific** Σ_k

2. **Log of the Gaussian Density**. Taking the **log** of this expression (just like in LDA, page 78), we get a **Discriminant Function**:

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \cdot \log |\Sigma_k| - \frac{1}{2} \cdot \left(\mathbf{x} - \mu_k\right)^T \Sigma_k^{-1} \left(\mathbf{x} - \mu_k\right) + \log\left(\pi_k\right) \qquad (58)$$

The **quadratic part** comes from:

$$\left(\mathbf{x} - \mu_k\right)^T \Sigma_k^{-1} \left(\mathbf{x} - \mu_k\right)$$

Which is a **quadratic form**. A **Quadratic Form** is just a fancy way to describe any expression where variables are multiplied together, especially with squares and cross-terms.

The **decision boundary** between class k and l is:

$$\delta_k\left(\mathbf{x}\right) = \delta_l\left(\mathbf{x}\right)$$

This becomes a **quadratic equation** in $\mathbf{x}$. Hence, the boundary is a **curve**, not a straight line (in 2D curved lines, in 3D curved surfaces, and in p-D quadratic hypersurfaces).

✅ When to Prefer QDA Over LDA

✔ Class Covariances Are Clearly Different. If our data shows that:

- Class A is tightly clustered.
- Class B is widely spread out.
- Or they have very different **orientations**.

Then QDA is the better choice because it allows:

$$\Sigma_1 \neq \Sigma_2 \neq \ldots \neq \Sigma_K$$

This leads to **more accurate boundaries**, especially when classes have **non-linear separation**.

✔ We Have Enough Data. QDA estimates **a full covariance matrix for each class**, which means:

- More parameters to estimate.
- Higher risk of overfitting if sample size is small

So we **use QDA when our dataset is large enough** to support all those estimates reliably.

✔ We Suspect Curved Boundaries. If we visually or intuitively suspect:

1. The data cannot be separated with a straight line.
2. There's curvature in how the classes divide.

Then QDA is more likely to model the real structure.

⚠ When to Avoid QDA

✖ Small sample size. Estimating separate covariance matrices can become unstable.

✖ High-dimensional data. In p-dimensional space, each Σ_k has $\dfrac{p\,(p+1)}{2}$ parameters, QDA becomes data-hungry.

✖ Risk of overfitting. More flexibility, then higher variance, especially with noise or unbalanced classes.

Use **LDA** when...	Use **QDA** when...
We want simplicity and robustness	We want flexibility and accuracy
Covariances are similar	Covariances are different
Data is limited	We have enough samples per class
Boundaries are approximately linear	Boundaries appear curved

Example 4: Overfitting
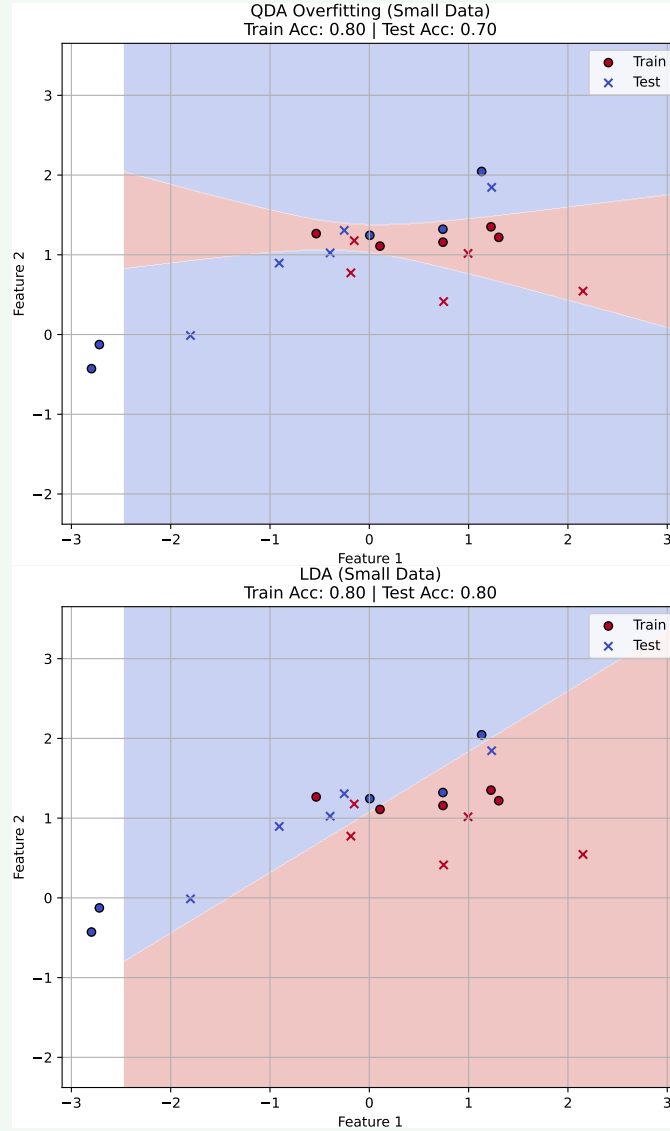
In this example, both QDA and LDA were trained on a very small dataset with only 10 training samples per class. QDA achieved 80% accuracy on the training set by fitting a flexible, curved decision boundary that tightly adapted to the few observed points. However, this resulted in poor generalization, with test accuracy dropping to just 70%, a possible case of overfitting. In contrast, LDA used a simpler, linear decision boundary, achieving same training accuracy (80%) but significantly better performance on the test set (80%). This demonstrates that LDA, being more constrained, generalizes better in low-data regimes, whereas QDA requires more data to avoid overfitting its more complex model.

3.5   Comparison LDA vs QDA

✖ Core Similarities

Aspect	Description
Both are generative	They model how each class generates data using a multivariate Gaussian.
Both use Bayes' rule	Classification is based on computing posterior probabilities.
Both are supervised	They require labeled training data.
Both work for multiple classes	Not limited to binary classification.

⚖ Key Differences

Feature	LDA	QDA
Covariance assumption	Same for all classes: Σ	Unique for each class: Σ_k
Decision boundary	Linear (straight lines/planes)	Quadratic (curved boundaries)
Model complexity	Fewer parameters, then simpler	More parameters, then more flexible
Training data needed	Works well with small datasets	Needs more data (per class)
Risk of overfitting	Lower	Higher
Bias-Variance tradeoff	Higher bias, lower variance	Lower bias, higher variance
Shape of decision region	All classes have same shape	Each class can have different shape/size/orientation

✅ Rule of Thumb

Situation	Recommended
We want a **stable model** with limited data	Use **LDA**
We have **enough data** and suspect curved decision boundaries	Try **QDA**
Our classes look like **different clouds** (e.g. rotated, stretched)	Use **QDA**
Our classes are well-separated by a **straight line/plane**	**LDA** is likely sufficient

3.6　Bayes Classifier

3.6.1　What is the Bayes Classifier?

The **Bayes Classifier** is a **decision rule** that assigns each observation to the class with the **highest posterior probability**, based on the conditional probability distribution of the features given the class.

√ͯ **Formal Definition**

Let:

- $\mathbf{x} \in \mathbb{R}^p$ be a new observation (feature vector).

- $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$ be the possible **classes**.

- $\pi_k = P\left(Y = \mathcal{C}_k\right)$ be the **prior probability** of class k.

- $f_k(\mathbf{x}) = P\left(\mathbf{X} = \mathbf{x} \mid Y = \mathcal{C}_k\right)$ be the **class-conditional density**.

Then the **posterior probability** of class $\mathcal{C}_k$ given $\mathbf{x}$ is:

$$P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right) = \frac{\pi_k f_k\left(\mathbf{x}\right)}{\displaystyle\sum_{j=1}^{K} \pi_j f_j\left(\mathbf{x}\right)} \tag{59}$$

The **Bayes classifier** assigns $\mathbf{x}$ to the class with the **highest posterior**:

$$\hat{y} = \arg\max_k P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right) \tag{60}$$

✅ **Math Behind the Bayes Classifier**

We will try to explain it in the simplest terms possible. Imagine we're a doctor trying to decide if a patient has **flu** or **allergy**, based on their **temperature**. We know from past experience:

- People with the flu usually have higher temperatures.

- People with allergies usually have normal temperatures.

Now, a new patient walks in with a temperature of 37.8°C. We ask: "*given this temperature, what is the most likely diagnosis?*". The Bayes Classifier answer this by saying "*pick the class (flu or allergy) that is most likely, based on what we know about temperatures in each class.*". So:

- If flu patients often have temperatures around 37.8°C, and allergy patients don't;

- Then the Bayes Classifier says: "*assign this patient to the flu class.*".

Now that the intuition is clear, let's explain the math behind the Bayes Classifier.

We are trying to **classify a new observation x**, such as someone's temperature, into one of K classes (e.g., flu vs allergy). We want to choose the class $\mathcal{C}_k$ that **maximizes the probability that it's the correct class**, given the data. This is the **posterior probability** (page 77):

$$P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right)$$

To compute this, we use **Bayes' theorem**:

$$P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right) = \frac{\pi_k f_k\left(\mathbf{x}\right)}{\displaystyle\sum_{j=1}^{K} \pi_j f_j\left(\mathbf{x}\right)} \tag{61}$$

Let's explain each term:

- $\pi_k = P(Y = \mathcal{C}_k)$ is the **prior**, *how common is class k in general?* For example, if 40% of patients have the flu, then $\pi_{\text{flu}} = 0.4$.

- $f_k(\mathbf{x}) = P(\mathbf{X} = \mathbf{x} \mid Y = \mathcal{C}_k)$ is the **Class-conditional density**. For example, how likely is temperature 37.8°C *if* someone has the flu?

- The **numerator** $\pi_k f_k(\mathbf{x})$ tells us: how likely is it that someone is in class k and has this data $\mathbf{x}$?

- The **denominator** just makes sure that all the probabilities sum to 1.

Although the formula may be complex, we **don't need to compute the entire formula**. Since the denominator is the same for all classes, we only care about comparing:

$$\pi_k f_k\left(\mathbf{x}\right)$$

We choose the class k that **maximizes** this value:

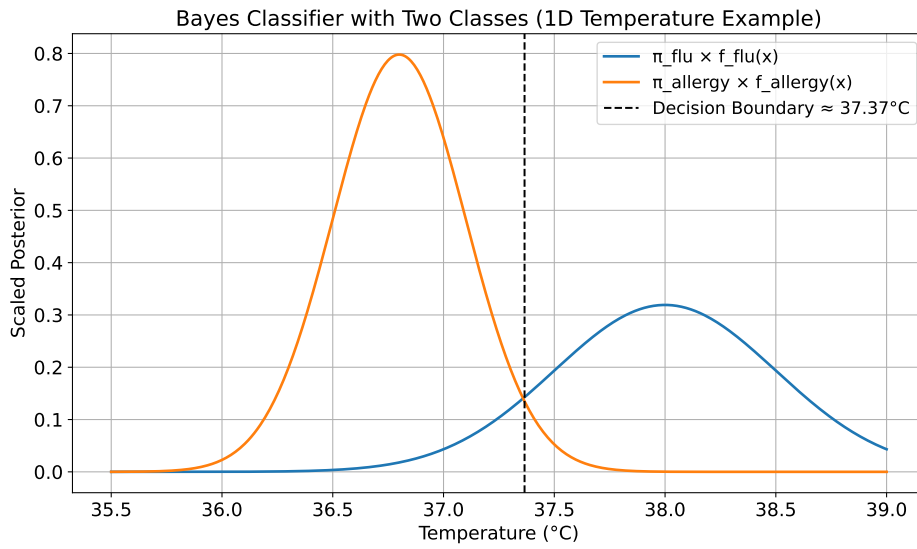$$\hat{y} = \arg\max_k \ \pi_k f_k\left(\mathbf{x}\right)$$

Figure 13: This plot shows how the Bayes classifier works in a simple case with two classes (flu and allergy), and one feature (body temperature). The blue curve $\pi_{\text{flu}} \cdot f_{\text{flu}}(x)$ is the likelihood of flu at each temperature, and the orange curve $\pi_{\text{allergy}} \cdot f_{\text{allergy}}(x)$ is the likelihood of allergy. At any temperature x, we compare the two curves; whichever is higher, we choose that class. The black dashed line shows the Bayes decision boundary, around 37.37°C: for $x < 37.37$ choose allergy, otherwise for $x > 37.37$ choose flu.

3.6.2 Bayes Error Rate

Even the **best possible classifier** (the Bayes classifier) can make mistakes. *Why?* Because **sometimes different classes overlap**: for some values of $\mathbf{x}$, it's just not clear which class it came from, even with perfect knowledge of the distributions. This unavoidable error is called the **Bayes Error Rate**.

The **Bayes Error Rate** is the **lowest possible classification error achievable by any classifier**. It's the error made by the Bayes classifier, which is optimal. Mathematically, it's the probability that the Bayes classifier **predicts the wrong class**:

$$\text{Bayes Error Rate} = \mathbb{E}_{\mathbf{X}}\left[1 - \max_{k} P\left(Y = \mathcal{C}_k \mid \mathbf{X} = \mathbf{x}\right) \right] \tag{62}$$

This means:

- For each observation $\mathbf{x}$, compute the **posterior probabilities** for each class.

- Keep the **highest one**, that's our confidence in the prediction.

- Subtract it from 1, that's the chance of being wrong at that point.

- Average this over all possible $\mathbf{x}$, gives us the total Bayes error.

📖 Binary Case (2 classes)

Since the previous formula works for any number of classes, *why focus on the binary case?* Because in the 2-class case, everything simplifies nicely:

- We only need to compare two posterior probabilities.

- The error at each $\mathbf{x}$ is just the smaller of the two posteriors.

- The total Bayes error becomes the integral of $\min\left(\pi_1 f_1\left(x\right), \pi_2 f_2\left(x\right)\right)$, which is easy to compute or visualize.

So the binary case gives a clean formula and is ideal for examples.

For two classes $\mathcal{C}_1$ and $\mathcal{C}_2$, the error at any point $\mathbf{x}$ is:

$$\text{Error at } \mathbf{x} = \min\left(P\left(Y = \mathcal{C}_1 \mid \mathbf{x}\right),\ P\left(Y = \mathcal{C}_2 \mid \mathbf{x}\right)\right) \tag{63}$$

So:

$$\text{Bayes Error Rate} = \int \min\left(\pi_1 f_1\left(\mathbf{x}\right), \pi_2 f_2\left(\mathbf{x}\right)\right) \mathrm{d}\mathbf{x} \tag{64}$$

That is: **integrate over the overlap** of the two class distributions, where the "wrong" class is more likely.

- If the class distributions are **well separated**, the Bayes error is **very low** or even 0.

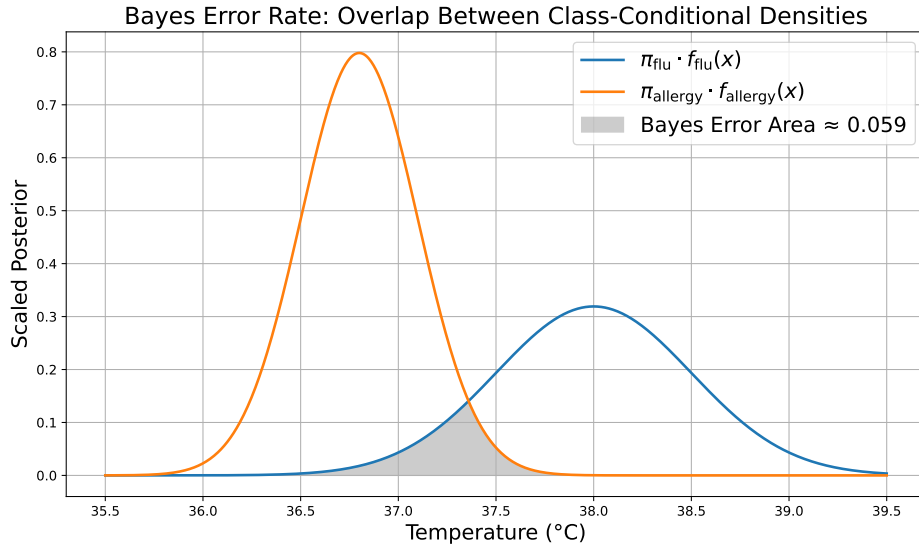- If they **overlap**, the Bayes classifier will still make **errors**, no matter how our model is.

Figure 14: This plot shows the Bayes error rate visually as the gray shaded area. The value (from the integral of the shaded region) is ≈ 0.059, meaning even the best classifier will make an error 5.9% of the time in this setting.
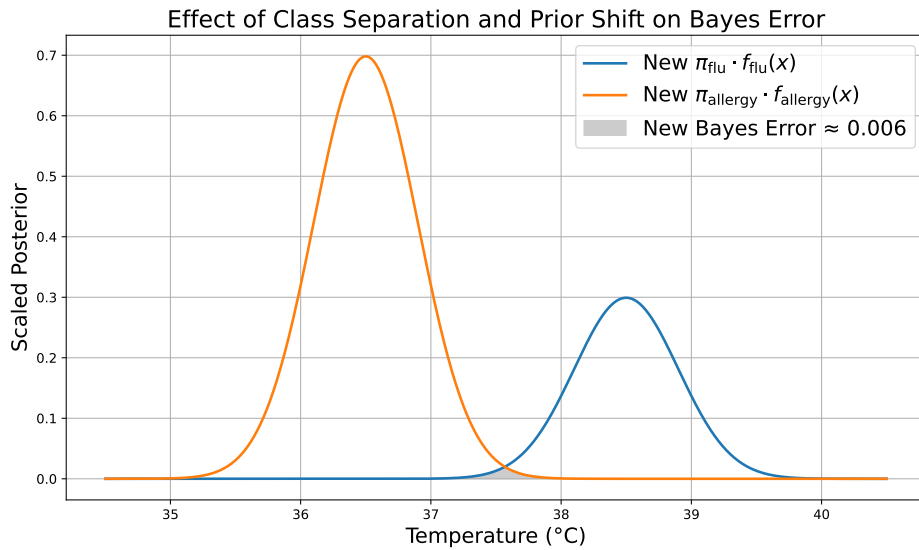

Figure 15: This plot is similar to the previous one, but here we increase the separation between the class means and change the priors. The two curves overlap much less, then the shaded area is smaller. The Bayes error rate drops significantly (now around 0.006, 0.6%) compared to $\approx 5.9\%$ before.

🕑 Performance Measures

First, it's important to know that the **Oracle Classifier is another name for the Bayes classifier**. It emphasizes that the classifier knows the true class-conditional distributions and gives the lowest possible error.

In simulation studies or theoretical analysis, we can:

- Generate data from known distributions;

- Then simulate the Bayes/Oracle classifier (since we know f_k);

- And evaluate its **true error rate**.

This gives us a **reference point** to compare real classifiers like LDA, QDA, or logistic regression.

Once we have a classifier (e.g., LDA, QDA, Bayes), we **need to evaluate how well it performs**. We use two key error measures:

- **Apparent Error Rate (APER)**. It is the **training error**. It's calculated by applying the classifier to the **same data it was trained on**. Often **optimistic**, it <mark>underestimates the real error</mark>, especially if the model is overfitting.

$$\text{APER} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathbb{I}\left(\hat{y}_i \neq y_i\right) \quad \text{(on training data)} \tag{65}$$

- **Actual Error Rate (AER)** (or **True Error Rate**). The **true classification error**, measured on **independent test data**. Gives a <mark>more honest estimate of model performance</mark>. In simulations, we can also compare to the **true labels generated from known distributions**.

$$\text{AER} = \frac{1}{m} \cdot \sum_{j=1}^{m} \mathbb{I}\left(\hat{y}_j \neq y_j\right) \quad \text{(on test data or true labels)} \tag{66}$$

3.6.3 Bayes vs LDA/QDA

As we've seen in previous sections, the **Bayes classifier** is the gold standard because it yields the **lowest possible error** <mark>when the true distributions of the data are known</mark>.

But in practice, we rarely know those distributions. That's where LDA (Linear Discriminant Analysis) and QDA (Quadratic Discriminant Analysis) come in. They are **approximations** of the Bayes classifier under some assumptions.

As we've seen, the Bayes classifier is:

$$\hat{y} = \arg\max_k \ \pi_k \cdot f_k\left(\mathbf{x}\right)$$

Where $f_k\left(\mathbf{x}\right)$ is the **true class-conditional density that is not known in practice** because we only have data. <mark>**That's why we can't use the Bayes classifier in practice, it's only a theoretical benchmark.**</mark> Instead, LDA and QDA assume that the shape of f_k is known. However, they do not know the shape of f_k; they estimate it from the data.

- <mark>LDA</mark>: **assumes all classes share the same covariance matrix**.
  📗 **Assumptions**

  – Class-conditional distributions: Multivariate Gaussian.
  – **Same covariance matrix** Σ for all classes.
  – Different means μ_k.

  This leads to a **linear decision boundary**. It is linear because when we plug the Gaussian formula into the Bayes rule and simplify under equal Σ, we get a linear function of $\mathbf{x}$.

- <mark>QDA</mark>: **allows each class to have its own covariance matrix**.
  📗 **Assumptions**

  – Class-conditional distributions: Multivariate Gaussian.
  – **Different covariance matrix** Σ_k for each class.

  This leads to a **quadratic decision boundary**.

Property	Bayes Classifier	LDA	QDA
Requires true f_k	✔ Yes	✘ No, assumes Gaussian	✘ No, assumes Gaussian
Covariance	Anything	Shared across classes	One per class
Decision boundary	Arbitrary (optimal)	Linear	Quadratic
Flexibility	Highest	Low	Medium
Risk of overfitting	None (theoretical)	Low	Higher
When to use	Only in theory	When classes are well-separated, simple	When boundaries are curved, and more data is available

❓ What Is the Purpose of the Bayes Classifier?

Although the Bayes classifier cannot be used in practice, it plays a fundamental role in machine learning and statistics:

- ✔ **It Defines the Best Possible Classifier**. It is the **gold standard**: the classifier with the **lowest possible error rate**. This error is called the Bayes error rate. All other classifiers are trying to get as close as possible to this theoretical optimum.

- ✔ **It Helps Us Understand the Role of Assumptions**. By comparing real classifiers to the Bayes rule, we can say:

  - *How much error is due to* ***model approximation?***
  - *How much is due to* ***data limitations?***

  For example:

  - LDA ≈ Bayes **only** when its assumptions are correct (Gaussian, equal covariance).
  - When assumptions are wrong (e.g., non-Gaussian data), LDA will do worse than Bayes.

- ✔ **It Provides a Foundation for Deriving Other Classifiers**. Many practical classifiers (LDA, QDA, Naive Bayes, even some neural networks) are **inspired by Bayes' rule**. They make simplifying assumptions or estimate f_k, but the **form of the classifier** is based on Bayes.

- ✔ **It Lets Us Theoretically Analyze the Limits of Learning**. It helps define concepts like bias-variance tradeoff, irreducible error, and Bayes consistency. We can measure how "good" a classifier is by how close it comes from to **Bayes-optimal**.

Example 5: Bayes Analogy

Think of the Bayes classifier like a **perfect GPS** that always knows the best route, because it sees the **entire traffic map of the world**. In practice, we only get local signals (data), so our actual GPS (LDA, QDA, etc.) is trying to guess the best route, and we measure its performance **by how close it is to the perfect one**.

4   Linear Regression

4.1   Regression vs Classification

In the previous chapter, we examined classification methods such as the Bayes classifier, QDA, and LDA. Now, we will examine regression methods.

- **Classification**

    - ✅ **Goal**: predict a **category or class label**.
    - ❓ **Output**: variable $Y_i \in \{1, 2, \ldots, G\}$.
    - √ˣ **Function**
    $$f : \mathcal{X} \to \{1, 2, \ldots, G\}$$

      For example, email spam detection (spam, not spam), disease diagnosis (positive, negative).

- **Regression**

    - ✅ **Goal**: predict a **continuous numerical value**.
    - ❓ **Output**: variable $Y_i \in \mathbb{R}$.
    - √ˣ **Function**
    $$f : \mathcal{X} \to \mathbb{R}$$

      For example, predicting house prices, forecasting temperature, estimating blood pressure.

Feature	Regression	Classification
Output type	Continuous	Categorical (Discrete labels)
Loss function	Usually MSE: $(y - \hat{y})^2$	Usually 0-1 loss or log-loss
Model types	Linear regression, Ridge, etc.	Logistic regression, SVM, etc.
Evaluation	RMSE, R^2	Accuracy, Precision, AUC

Table 8: Main Differences.

4.2   Simple Linear Regression (SLR)

4.2.1   Model Formulation and OLS Estimation

The **Simple Linear Regression (SLR)** is a **statistical method** that models the relationship between one **independent variable** (also called predictor or input) X and one **dependent variable** (also called response or output) Y, using a **straight line**. The model assumes:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \tag{67}$$

Where:

- Y_i: **observed response** (output) for sample i. It is the value we want to predict.

- X_i: **predictor** (input). It is the value we use to predict.

- β_0: **intercept** (value of Y when $X = 0$).

- β_1: **slope** (change in Y for a unit increase in X). It indicates how much Y increases when X increases by 1.

- ε_i: **random error** (assumed to be $\sim \mathcal{N}\left(0, \sigma^2\right)$, in other words normally distributed).

It can also be viewed in matrix form:

$$\mathbf{Y} = X\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

It's called **simple** because there is only one predictor (X) and if there were multiple predictors $X_1, X_2, \ldots$, we would call it **multiple linear regression**.

✅ **Goal**. We want to **predict a continuous outcome** $Y \in \mathbb{R}$ based on a single input variable $X \in \mathbb{R}$ using a **linear relationship**. For example, predicting house price Y based on house size X.

✅ **Key Idea**

We want to find the **best straight line**:

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

That minimizes the **total squared error** between the actual Y and the predicted $\hat{Y}$.

Figure 16: This is an example of an SLR model in action. The X-axis represents the independent variable, hours studied, the Y-axis represents the dependent variable, exam score, the orange dots represent the actual observed data points, and the black line represents the best-fit regression line $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$. The OSL finds the straight line that best represents the trend in the data; here, students who study more tend to get higher exam scores. The line summarize that linear relationship with just 2 numbers: intercept and slope.

❷ **What is OSL Estimation (Ordinary Least Squares)?**

Ordinary Least Squares Estimation (OSL Estimation) is a method **to find the best-fitting line** for our data by **minimizing the total squared errors** between observed values and predicted values.

Given the SLR model, we don't know β_0 and β_1. We want to **estimate them** using data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. We define the prediction as:

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

And the **error (residual)** for point i is:

$$e_i = Y_i - \hat{Y}_i \tag{68}$$

❷ **What does OLS Minimize?**

OLS finds $\hat{\beta}_0$ and $\hat{\beta}_1$ that **minimize the sum of squared residuals**:

$$\text{RSS} = \sum_{i=1}^{n} (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2 \tag{69}$$

This is called the **Residual Sum of Squares (RSS)**.

📗 Core Idea of RSS

We want our regression line to be **as close as possible** to the observed data points. But *how do we measure "closeness"?* By the **vertical distance** between each true point (X_i, Y_i) and the predicted point $(X_i, \hat{Y}_i)$, that is:

$$e_i = Y_i - \hat{Y}_i \quad \text{(residual)}$$

But some residuals are negative, some positive. So **we square them to avoid cancellation and emphasize large errors**:

$$\text{RSS} = \sum e_i^2$$

Figure 17: In this plot, we highlight the residual and the residual sum of squares (RSS). The blue dots are the actual data point (X_i, Y_i), the red line is the regression line $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$, the gray dashed lines are the residuals $e_i = Y_i - \hat{Y}_i$, i.e., vertical distances from each point to the line, and the RSS value is displayed in the title, it's the sum of the squares of these residuals.

OLS chooses the red line that **minimizes the total gray distances squared (RSS)**. A different line (e.g., tilted more or less) would **increase the sum of squared errors** (see Figure 18, page 104). This is **why** OLS minimizes RSS, it leads to the line that best balances all errors.
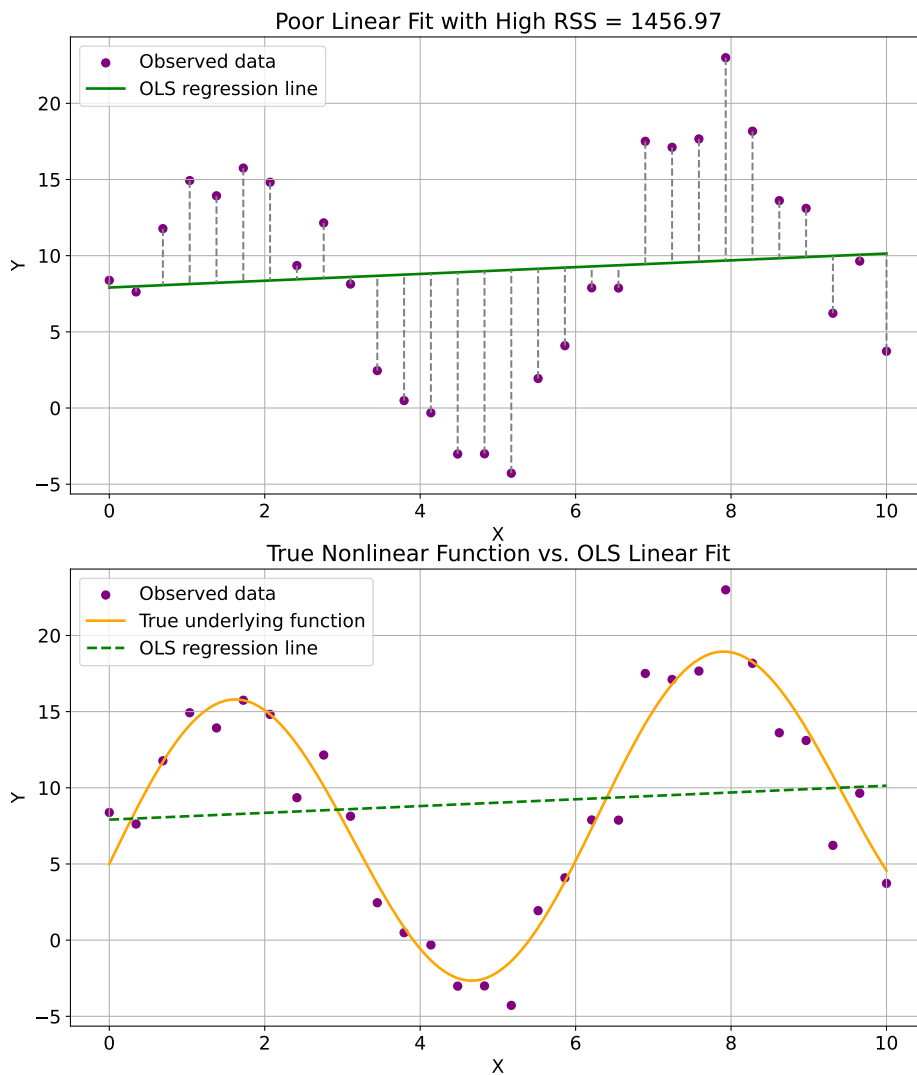
Figure 18: This is an example of a poor linear fit, which is also highlighted by the RSS. The purple dots are the true data points, which follow a **nonlinear (sinusoidal) trend**, green line is the best-fitting **straight line** found by OLS, the gray dashed lines are the **residuals** (the vertical errors between data points and the regression line), and the RSS is very high because the straight line fails to capture the curve in the data.

OLS always minimizes the **sum of squared residuals**, but if the model is misspecified (e.g., fitting a line to curved data), the **RSS will still be high**.

✅ Good fit ⇒ **low RSS**

❌ Bad fit ⇒ **high RSS**

OLS chooses the line that makes RSS as small as possible, even if the fit is poor due to model limitations.

📑 Ordinary Least Squares (OLS) Estimators

Solving for the minimum yields two formulas for the best-fitting line, called OLS estimators. **Ordinary Least Squares estimators** are the **values** of $\hat{\beta}_0$ (intercept) and $\hat{\beta}_1$ (slope) **that minimize the Residual Sum of Squares (RSS)** in simple linear regression (SLR):

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

We want to find $\hat{\beta}_0$, $\hat{\beta}_1$ such that:

$$\text{RSS} = \sum_{i=1}^{n}(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2$$

Is <mark>minimized</mark>.

OLS gives **explicit formulas** for the best values:

- **Slope**

$$\hat{\beta}_1 = \frac{\displaystyle\sum_{i=1}^{n}\left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\displaystyle\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2} \tag{70}$$

  Measures how much Y changes for a 1-unit increase in X. It's the **slope** of the best-fitting line.

- **Intercept**

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X} \tag{71}$$

  Value of Y when $X = 0$.

These two formulas define the **Regression Line**:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X \tag{72}$$

For example, suppose we compute $\hat{\beta}_0 = 3$, $\hat{\beta}_1 = 2$, then our model says: "*each time X increases by 1, Y increases by 2. If $X = 0$, predicted $Y = 3$*".

4.2.2   Geometric Interpretation of Linear Regression

Understanding OLS **geometrically** gives deep intuition about what's really happening during the model fitting process.

⚒ **Setup.** Suppose we have a dataset:

- We observe n responses and save them in a vector of observed responses $\mathbf{Y} \in \mathbb{R}^n$:

$$\mathbf{Y} = [Y_1, Y_2, \ldots, Y_n]^T$$

- We have one predictor X, but the model includes a constant (intercept) too. So the **design matrix** $\mathbf{X} \in \mathbb{R}^{n \times 2}$ has:

$$\mathbf{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix}$$

  The column of ones is fundamental because, to calculate the regression line $\hat{\mathbf{Y}}$ in matrix form, we need to $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$, which translates to:

$$\hat{\mathbf{Y}} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \boldsymbol{\varepsilon} = \begin{bmatrix} \beta_0 + \beta_1 X_1 \\ \beta_0 + \beta_1 X_2 \\ \vdots + \vdots \\ \beta_0 + \beta_1 X_n \end{bmatrix} + \boldsymbol{\varepsilon}$$

  So the column of ones ensures the **intercept** β_0 is part of every prediction. Without the column of ones, we would be **forcing the line to pass through the origin**, i.e., no intercept!

Now, the column space of $\mathbf{X}$ is all combinations of the two vectors: the all-ones vector (for the intercept), and the values of X:

$$\mathrm{Col}\,(\mathbf{X}) = \{\beta_0 \cdot \mathbf{1} + \beta_1 \cdot X\}$$

This is a **2D plane inside** $\mathbb{R}^n$.

✅ **Goal.** The goal of OLS is to **project** the vector $\mathbf{Y}$ onto the **space spanned by the predictors** in $\mathbf{X}$. In other words, since it is impossible to fit all our observed values ($\mathbf{Y}$) perfectly with a line, **OLS selects a vector $\hat{\mathbf{Y}}$ in the column space of $\mathbf{X}$ that is as close as possible to $\mathbf{Y}$**. This is done by <mark>orthogonal projection</mark>:

$$
\begin{aligned}
\hat{\mathbf{Y}} \;&=\; \text{projection of } \mathbf{Y} \text{ onto the space spanned by the columns of } \mathbf{X} \\
&=\; \mathbf{X}\hat{\beta} \\
&=\; \text{projection of } \mathbf{Y} \text{ onto } \mathrm{Col}\,(\mathbf{X})
\end{aligned}
$$

It is important because the **projection minimizes the distance** from $\mathbf{Y}$ (our observed values) to $\hat{\mathbf{Y}}$ (regression line to find), exactly what OLS does.
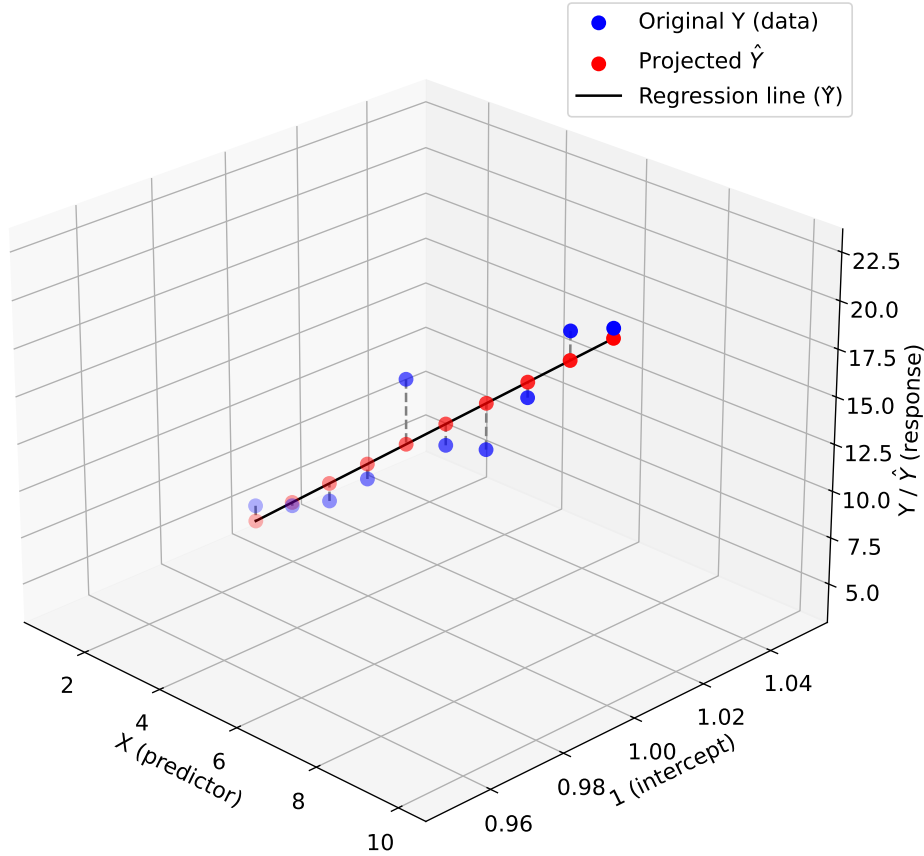
Figure 19: Blue dots are the actual observed Y_i values, the full response vector **Y**. The red dots are the projected values $\hat{Y}_i$, they lie in the space spanned by the columns of **X**. The gray dashed lines are the residuals, the vertical distance from each observed Y_i to its projected $\hat{Y}_i$. The black line is the fitted regression line (through the red points), this lies in the **column space** of the design matrix.

The design matrix **X** has two columns: one column of ones (to include the intercept) and one column with the values of X. The red points from the best approximation $\hat{\mathbf{Y}} = \mathbf{X}\hat{\beta}$. The black line shows the **vector space of all possible** $\hat{Y}$ values using linear combinations of the intercept and X. **OLS chooses** the projection of **Y onto that line**, minimizing the **sum of squared residuals**.

√× Vector Space View

In Simple Linear Regression, the design matrix is, as we have seen, as follows:

$$\mathbf{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix}$$

OLS finds the **orthogonal projection** of $\mathbf{Y}$ onto this space:

$$\hat{\mathbf{Y}} = \mathbf{P}_X \mathbf{Y} \tag{73}$$

Where $\mathbf{P}_X$ is called **Projection Matrix**, or **Hat Matrix** because it maps $\mathbf{Y}$ to its fitted values $\hat{\mathbf{Y}}$. It is defined as follows:

$$\mathbf{P}_X = \mathbf{X} \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \tag{74}$$

It **projects** any vector in $\mathbb{R}^n$ (like $\mathbf{Y}$) onto the **column space of $\mathbf{X}$**, that is, onto the space of all possible predicted values.

Finally, the residual vector is:

$$\mathbf{e} = \mathbf{Y} - \hat{\mathbf{Y}} \tag{75}$$

That is **orthogonal** to the fitted vector $\hat{\mathbf{Y}}$.

4.3   Model Evaluation

After fitting the regression line, the next question is: *how good is the model at explaining the data?* That's what this section is about.

✅ Goal of Model Evaluation

We want to understand:

1. How much of the variation in Y is **explained** by the model?

2. How much remains **unexplained** (random noise or poor fit)?

This leads to the **decomposition of variance** and the **coefficient of determination** R^2.

📗 Variance Decomposition

We break down the total variability in the output Y into:

- **Regression Sum of Squares (SSR)** (or explained sum or squares)

$$\text{SSR} = \sum_{i=1}^{n} \left(\hat{Y}_i - \bar{Y} \right)^2 \tag{76}$$

  Measures how much of the variability in Y is **explained by the model**. It's the variability **captured by the regression line**.

- **Residual Sum of Squares (RSS)** (or error sum of squares)

$$\text{RSS} = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i \right)^2$$

  Measures the remaining **unexplained error**, how far off the predictions are from the actual values. This was explained on page 102.

- **Total Sum of Squares (TSS)**

$$\text{TSS} = \sum_{i=1}^{n} \left(Y_i - \bar{Y} \right)^2 \tag{77}$$

  Measures the **total variability** in the data. It's how much the actual values Y_i vary around the mean $\bar{Y}$. In other words, how "*spread out*" the data is.

  In Ordinary Least Squares (OLS) regression, we always have the following **Fundamental Identity**:

$$\text{TSS} = \text{SSR} + \text{RSS} \tag{78}$$

  OLS finds $\hat{\beta}$ to **minimize RSS**, and it guarantees that the residuals $\mathbf{e} = Y - \hat{Y}$ are **orthogonal** to the fitted values. Because of this orthogonality, the squared norms satisfy the Pythagorean theorem:

$$\left\| Y - \bar{Y} \right\|^2 = \left\| \hat{Y} - \bar{Y} \right\|^2 + \left\| Y - \hat{Y} \right\|^2 \quad \Rightarrow \quad \text{TSS} = \text{SSR} + \text{RSS}$$

📗 Coefficient of Determination

The **Coefficient of Determination** (or simply R^2, or **R-Squared**) is **one of the most important metrics** to evaluate how well our linear regression model fits the data. It is defined as:

$$R^2 = \frac{\text{Explained Variation}}{\text{Total Variation}} = \frac{\text{SSR}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} \tag{79}$$

It tells us the **proportion** of the variance in Y that is **explained by our regression model**.

> ✅ If our model predicts the values **perfectly**, then:
>
> $$\hat{Y}_i = Y_i \quad \Rightarrow \quad \text{RSS} = 0 \quad \Rightarrow \quad R^2 = 1$$

> ⚖️ If our model does **not better than predicting the mean**, then:
>
> $$\hat{Y}_i = \bar{Y} \quad \Rightarrow \quad \text{SSR} = 0 \quad \Rightarrow \quad R^2 = 0$$

> ❌ If our model is **worse than predicting the mean**, then:
>
> $$\text{RSS} > \text{TSS} \quad \Rightarrow \quad R^2 < 0$$

For example, if TSS $= 100$, and RSS $= 20$, then:

$$R^2 = 1 - \frac{20}{100} = 0.80 \quad \Rightarrow \quad \text{Our model explains 80\% of the variation in } Y$$

R^2 always **increases** as we add **more predictors** (even useless ones), and for multiple regression, we often use Adjusted R^2 to penalize unnecessary complexity.

Figure 20: A side-by-side comparison of a Good Model vs. a Bad Model using the same dataset.

The coefficient of determination, R^2, is a widely used metric to assess model fit. However, it tends to increase with model complexity, even when additional predictors do not improve the model's true explanatory power, leading to overfitting. To address this, a more robust variant, the Adjusted R^2, was developed (see page 152). It is particularly useful in model selection, as it penalizes unnecessary complexity and provides a more reliable assessment of model performance.

Metric	What it means	Good Model	Bad Model
TSS	Total variability in Y	3821.1	3821.1
		Same for both, this is fixed by the data	
RSS	Unexplained error	325.96	3751.18
		Low, model fits well	High, model fits poorly
SSR	Explained variation	3495.13	69.92
		High, captures most of the structure	Very low
R^2	Proportion of explained variance	0.91	0.02
		91% explained	Only 2% explained

Table 9: Model evaluation of Figure 20.

This illustrates how R^2 measures **how well ur model captures the true pattern in the data**.

4.4   Statistical Inference

Statistical Inference is the process of using data from a sample to learn something about the **true population parameters**, and to **quantify uncertainty**.

In linear regression, it means: we've estimated $\hat{\beta}_0$, $\hat{\beta}_1$ from a sample, but the true values β_0, β_1 are unknown. So we ask:

- *Are these estimates reliable?*

- *Are they statistically significant?*

- *How much could they vary if we repeated the data collection?*

❓ **So what do we do in regression inference?**   We perform tests and intervals on the model coefficients:

- **Sampling Distributions & CLT** (page 113). Under the hood, regression estimates come from random samples. The **Central Limit Theorem** (CLT) tells us these **estimates** follow an **approximate normal distribution**. This gives us the **foundation** to build confidence intervals and test hypotheses.

- **t-Test for Individual Coefficients** (page 115). We test each $\hat{\beta}_j$ (like slope or intercept) against 0: *is this variable useful?* If $\hat{\beta}_j$ is far from 0 (statistically), we say it's **significant**.

- **Global F-Test** (page 116). This is a test on the **entire model**: *does any variable in the regression help explain the response?* It compares the model with predictors vs. a model with just the intercept.

Question	Answered by
Is this predictor significant?	t-test
Is the model better than nothing?	F-test
How much do my estimates vary?	CLT and confidence intervals

4.4.1   Sampling Distributions & Central Limit Theorem (CLT)

❷ Why do we care?

In the real world we don't know the true coefficients β_0, β_1. We **estimate them** from a sample data. But if we collected **a different sample**, we'd get **different estimates**.

So the question is: *How variable are the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$?* In other words: *"if we repeated the data collection and regression **many times**, how much would $\hat{\beta}_1$ change from sample to sample?"*. This is where **sampling distributions** and the **Central Limit Theorem (CLT)** come in.

❷ What is a Sampling Distribution?

A **Sampling Distribution** is the **distribution of a statistic** (like $\hat{\beta}_1$) that we get when we **repeat the sampling process many times**.

Imagine we have a population where:

$$Y = 3 + 2X + \varepsilon$$

We randomly collect 50 data points from this population, and we get $\hat{\beta}_1 = 1.96$. We collect another 50 data points from the same process, and now we get $\hat{\beta}_1 = 2.15$.

Each time we get a different dataset (due to random noise), so we get a different estimate of $\hat{\beta}_1$. Now image we plot a histogram of all those $\hat{\beta}_1$ values. That histogram is the **sampling distribution** of $\hat{\beta}_1$.

Sampling Distribution of $\hat{\beta}_1$ (Slope)
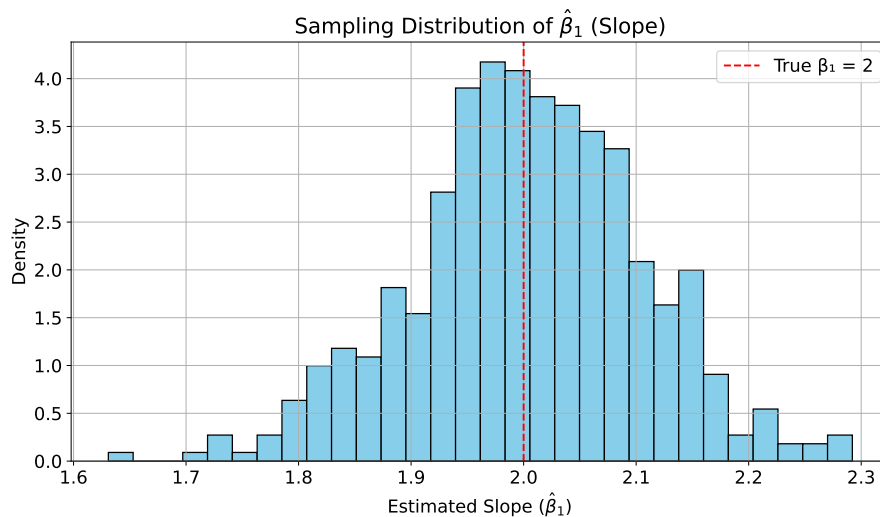
📗 What is the Central Limit Theorem (CLT)?

If we take **many random samples** from any population (no matter its shape), and compute the **mean** (or other statistics) for each sample, then the **sampling distribution** of that statistic will be **approximately normal**, as long as the sample size is large enough.

Theorem 1 (**Central Limit Theorem**). *Let X_1, X_2, ..., X_n be independent random variables with:*

- *Any distribution (not necessarily normal)*

- *Mean μ and finite variance σ^2*

Then the sample mean:

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i$$

*Has a sampling distribution that **converges** to:*

$$\bar{X}_n \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right) \quad as\ n \to \infty$$

*In short, **means behave like normals**, even if the data itself doesn't!*

✅ Why does CLT matter in Regression?

Because it allows us to:

1. **Quantify Uncertainty**. When we run regression, we get estimates like $\hat{\beta}_1 = 2.1$. But it's just from one sample, *what if the true slope is different?* The CLT says: "if we took many samples, the estimates $\hat{\beta}_1$ would follow a normal distribution around the true value β_1". So now we know **how much those estimates vary**.

2. **Build Confidence Intervals**. CLT tells us we can say: "with 95% confidence, the true slope lies between 1.85 and 2.35". Without the CLT, we couldn't say that. We'd have no way to define what "95% confidence" even means.

3. **Perform Hypothesis Tests**. The CLT lets us test:

   - *Is the slope really different from zero?*
   - *Is this variable important in predicting the outcome?*

   That's done using the fact that $\hat{\beta}_1$ is **approximately normal**, thanks to the CLT.

4. **Trust the Shape of Our Estimators**. Even if our data isn't perfect (e.g., not normally distributed), CLT tells us: "*it's okay, our estimated slope is still behaving normally if our sample is large enough*".

The CLT gives us the theoretical foundation to measure uncertainty, build intervals, and test if our model is meaningful, even when we only have one dataset.

4.4.2   t-Test for Individual Coefficients

t-Test for Individual Coefficients is a **hypothesis test** to check whether an individual variable in our regression **is statistically significant**, i.e., whether it **truly affects the response variable** Y.

✅ Why do we do this?

We run a regression and get:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

But we want to know: "*is this slope $\hat{\beta}_1$ **actually meaningful**, or did we just get lucky with the data?*". That's exactly what the **t-test** answers.

🔧 How does it work?

The core logic behind the t-test is to determine **whether there is sufficient evidence to conclude that** $\beta_1 \neq 0$ in the population. In other words, we need to establish whether the estimated slope is significantly different from zero.

1. **Set up the hypotheses**.

   - Null Hypothesis H_0: $\beta_1 = 0$ (no effect)
   - Alternative Hypothesis H_1: $\beta_1 \neq 0$ (some effect)

2. **Compute the t-Statistic**. This tells us how many standard errors away is our slope from 0:

$$t = \frac{\hat{\beta}_1 - 0}{\text{SE}\left(\hat{\beta}_1\right)} \tag{80}$$

   It tells us **how far** our estimated slope $\hat{\beta}_1$ is from the null value (usually 0), in units of standard error.

3. **Compare to t-Distribution**. Use a t-distribution with $n - 2$ degrees of freedom. The **t-distribution** looks like a **normal distribution**, but it has fatter tails. We use the **t-distribution instead of the normal distribution** because we are **estimating the standard deviation from the data** rather than using the true population standard deviation.

   Finally, we compute the p-value. The **p-value** is the probability, **under the null hypothesis** H_0, of observing a t-statistic as extreme or more extreme than the one we got.

$$p = P\left(|T| > |t|\right) \quad \text{where } T \sim t_{n-2} \tag{81}$$

   This is calculated using the **cumulative distribution function (CDF)** of the t-distribution.

   and compute the p-value.

   - If p-value < 0.05, reject H_0
   - If p-value > 0.05, not significant

4.4.3   Global F-Test

While the t-test checks if an individual coefficient (e.g., β_1) is significant, the F-test asks: "*does the **entire regression model** explain the data **better than just using the mean**?*". In other words, do **any** of the predictors explain Y? Or are all the $\beta_j = 0$, meaning the model is useless?

✅ Hypotheses

- **Null Hypothesis H_0:**

$$\beta_1 = \beta_2 = \cdots = \beta_p = 0$$

  No predictors matter, model has no explanatory power.

- **Alternative Hypothesis H_1:**

$$\text{At least one } \beta_j \neq 0$$

  At least one predictor helps explain Y.

📋 How is the F-Statistic computed?

$$F = \frac{\dfrac{(\text{TSS} - \text{RSS})}{p}}{\dfrac{\text{RSS}}{(n - p - 1)}} = \frac{\text{Explained variance per predictor}}{\text{Unexplained variance per residual dof}} \tag{82}$$

Where:

- p is the number of predictors

- n is the number of observations

- TSS is the total sum of squares

- RSS is the residual sum of squares

We compare this F-statistic to an F-distribution with p and $n - p - 1$ degrees of freedom.

✅ What does it tell us?

- Large F, small p-value: the model is **statistically significant**.

- Small F, large p-value: the model doesn't explain the data better than a flat mean.

❓ Why use the Global F-Test?

Because it answers: "*is this model **useful at all**, or should we just predict Y using its average?*". This is especially useful when we have **multiple predictors**, or we want to test the **entire model at once**.
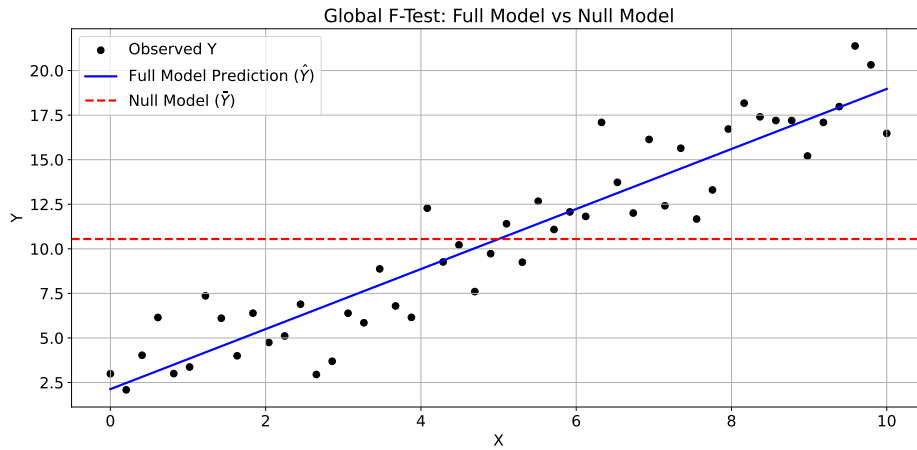
Figure 21: A visual explanation of the Global F-Test.

- Black points: the actual observed Y_i values

- Blue line: predictions from the **full regression model** $\left(\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i\right)$

- Red dashed line: prediction from the **null model**, simply the mean of Y, i.e., $\hat{Y}_i = \bar{Y}$

The **null model** (red line) says "Forget X, just guess everyone's output is the average". The **full model** (blue line) says "Use X to predict Y". The Global F-Test checks "does the blue line explain significantly more than the red one?".

$$F = \frac{\text{Improvement (Red} \rightarrow \text{Blue)}}{\text{Leftover Error}}$$

If:

- Blue is **much closer** to the points than red $\Rightarrow$ large F $\Rightarrow$ **significant model**.

- Blue is **barely better** than red $\Rightarrow$ small F $\Rightarrow$ **not useful**.

4.4.4 Summary

✅ **Goal**

After estimating the regression line:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

We want to answer:

- Is this model **trustworthy**?
- Are the coefficients **significantly different from 0**?
- What's the **uncertainty** in our predictions?

To do that, we use **Statistical Inference**.

1. Sampling Distributions & the Central Limit Theorem (CLT)

- Our coefficients $\hat{\beta}_0, \hat{\beta}_1$ come from a **sample**
- If we collected different data, we'd get different values
- The CLT tells us:

$$\hat{\beta}_j \sim \mathcal{N}(\beta_j, \text{Var}(\hat{\beta}_j)) \quad \text{(approximately)}$$

This normality enables confidence intervals and hypothesis testing.

2. Standard Errors and Confidence Intervals

- The **standard error (SE)** measures how much $\hat{\beta}_j$ would vary across samples
- A **95% confidence interval** for β_1 is:

$$\hat{\beta}_1 \pm t_{n-2} \cdot \text{SE}\left(\hat{\beta}_1\right)$$

- Interpretation: "*we're 95% confident the true slope lies in this range.*"

3. t-Test for Individual Coefficients

- Tests whether a specific β_j is **statistically significantly different from 0**
- Hypotheses:

$$H_0 : \beta_j = 0 \quad \text{vs} \quad H_1 : \beta_j \neq 0$$

- Test statistic:

$$t = \frac{\hat{\beta}_j - 0}{\text{SE}\left(\hat{\beta}_j\right)}$$

- Compare this to a **t-distribution** to compute the **p-value**

✅ If the p-value is small ⇒ **predictor matters**

❌ If the p-value is large ⇒ **no strong evidence**

📄 **4. Global F-Test (Model-Wide)**

- Tests whether **any** of the predictors have a significant effect

- Hypotheses:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0 \quad \text{(model does nothing)}$$

- F-statistic:

$$F = \frac{(\text{TSS} - \text{RSS})/p}{\text{RSS}/(n - p - 1)}$$

- Compare to an **F-distribution** with $(p, n - p - 1)$ degrees of freedom

✅ Small p-value ⇒ the model is **useful**

❌ Large p-value ⇒ the model might not be better than guessing the mean

Tool	Tests	Interpretation
CLT	All coefficients	Allows inference via normal approximation
SE + CI	Estimate accuracy	How much estimates vary
t-Test	Each coefficient	Is this predictor significant?
F-Test	Entire model	Is the model useful at all?

4.5   Uncertainty Intervals

After fitting a regression model, we often want to **predict** the response Y at a new value of X. But point predictions like:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

Only give a **single number**. That's not enough. In real life, we care about:

- **How uncertain is this prediction?**

- "*What range of outcomes should we expect?*"

✅ **Goal.** To build **intervals** around predictions that account for uncertainty. We study two types:

- **Confidence Interval (CI) for the Mean Response** (page 121). It calculates the range of the **average** Y value when the experiment is repeated many times at the same X value. In other words, it **captures uncertainty in the mean prediction**.

  Used when:

  - We're interested in the **mean trend**.
  - We're estimating the regression line itself.

- **Prediction Interval (PI) for a New Observation** (page 123). It calculates the range of a new Y value at a given X value, taking randomness into account. In other words, it **captures the uncertainty in individual outcome**.

  Used when:

  - We want to **predict an individual outcome**.
  - We care about real-world variability.

4.5.1   Confidence Interval (CI) for the Mean Response

Suppose we have fit a regression model:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

Now we pick a specific $X = x_0$, and ask: "*what's the range of values for the* **mean of** Y *at this* x_0?". This is the **Confidence Interval (CI) for the Mean Response** and the formula is:

$$\hat{Y}_0 \pm t_{n-2} \cdot \text{SE}\left(\hat{Y}_0\right) \tag{83}$$

Where:

- $\hat{Y}_0 = \hat{\beta}_0 + \hat{\beta}_1 x_0$ is the predicted mean.

- $\text{SE}\left(\hat{Y}_0\right)$ is the standard error of that prediction.

- t_{n-2} comes from a t-distribution with $n - 2$ degrees of freedom.

✅ **Why do we need it?**

Even if we know x_0, we don't know the true β_0, β_1, so our prediction $\hat{Y}_0$ is uncertain. This CI tells us: "*if we repeated this experiment many times and fit a new line each time, what range would the average Y fall in at x_0?*".

❓ **How to compute the Standard Error**

$$\text{SE}\left(\hat{Y}_0\right) = \sqrt{\hat{\sigma}^2 \left(\frac{1}{n} + \frac{(x_0 - \bar{X})^2}{\sum \left(X_i - \bar{X}\right)^2}\right)} \tag{84}$$

Where:

- $\hat{\sigma}^2$ is the MSE (mean squared error).

- The first term $\frac{1}{n}$ accounts for uncertainty in the intercept.

- The second term captures how far x_0 is from the center of the data.

Note: the CI is **narrowest at** $\bar{X}$ and gets wider as x_0 moves away.
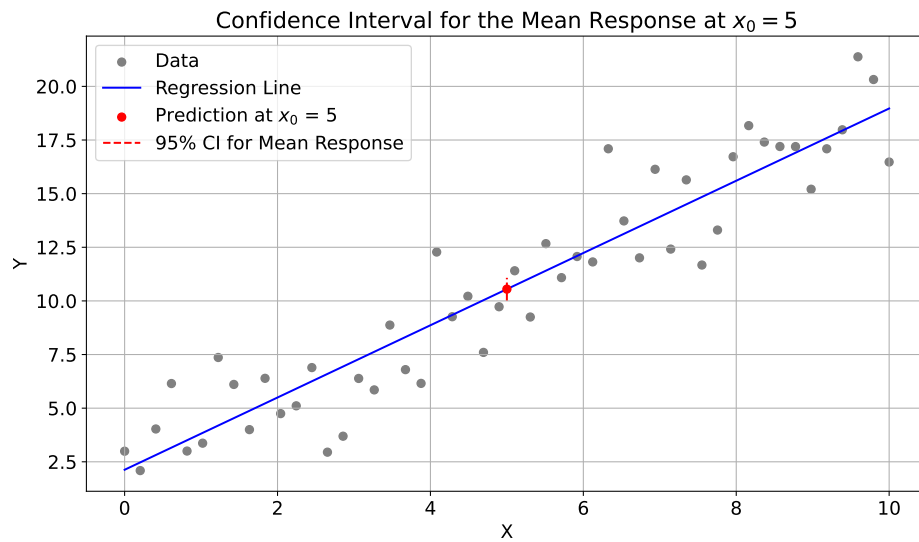
Figure 22: This plot shows a visual of the Confidence Interval (CI) for the Mean Response at $x_0 = 5$. The red dashed vertical line indicates the 95% confidence interval for the **average** Y at $x_0 = 5$, even if it is small. So, if we repeated this regression many times, we're 95% confident the average value of Y when $X = 5$ would fall in that vertical interval.

4.5.2   Prediction Interval (PI) for a New Observation

We've fit a regression model:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

Now we pick a specific value $X = x_0$ and ask: *"if we get a **new data point** with $X = x_0$, what range of values could the actual Y fall into?"*. That's the **Prediction Interval (PI) for a New Observation** and the formula is:

$$\hat{Y}_0 \pm t_{n-2} \cdot \mathrm{SE}_{\mathrm{pred}}(x_0) \tag{85}$$

Where:

$$\mathrm{SE}_{\mathrm{pred}}(x_0) = \sqrt{\hat{\sigma}^2 \left(1 + \frac{1}{n} + \frac{\left(x_0 - \bar{X}\right)^2}{\sum \left(X_i - \bar{X}\right)^2} \right)} \tag{86}$$

Compared to the Confidence Interval (page 121), this has an **extra +1** inside the square root, which accounts for the **random error in a single new observation**.

✅ Why do we need it?

Because real-world observations aren't always on the line, they **scatter around** due to noise ε. So we want a range that captures **both**:

- The **uncertainty in the estimated mean** at x_0, and

- The **random variation in a new individual outcome**

⚖️ Difference vs. Confidence Interval

Interval Type	Captures	Width
CI	Uncertainty in *average Y*	Narrower
PI	Uncertainty in *single Y*	**Wider**, includes noise!
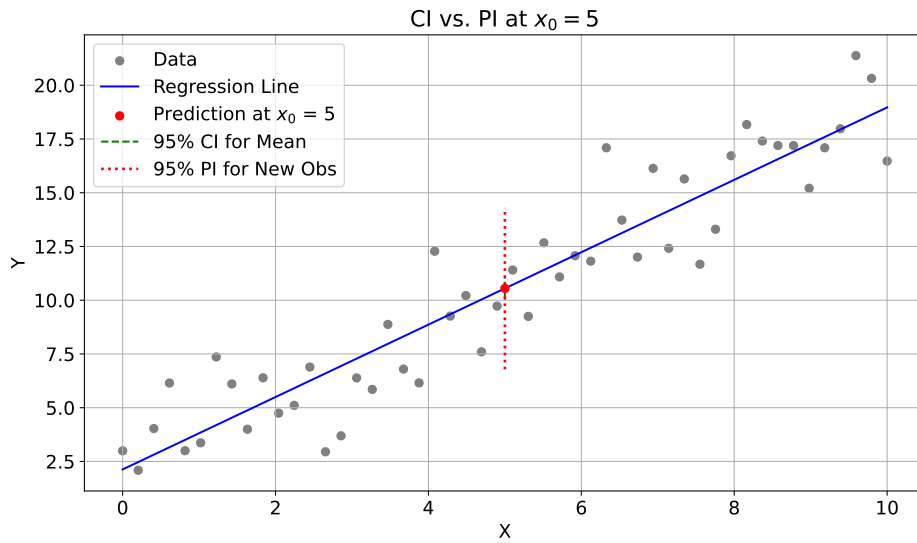
Figure 23: This plot illustrates the visual comparison of the two types of uncertainty interval at $x_0 = 5$. The red point is the predicted value $\hat{Y}_0$ at $x_0 = 5$.

Unlike Figure 22 on page 122, we also draw the prediction interval here.

- Green dashed line is the Confidence Interval (CI), for the mean of Y at $X = 5$. It is **narrow** because it only accounts for uncertainty in estimating the mean. It seems invisible because it is smaller than PI.

  The CI tells us: "*where would the **average** Y be at $X = 5$?*".

- Red dotted line is the Prediction Interval (PI), for a **new individual** with $X = 5$. It is **wider** because it also includes randomness (noise) in future observations.

  The PI tells us: "*where would a **single new** Y likely fall at $X = 5$?*".

4.6   Multiple Linear Regression (MLR)

❓ Why Move Beyond Simple Linear Regression?

Up until now, we've modeled a response variable Y as a function of **just one predictor** X:
$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$$

But in real-world problems, the outcome usually depends on **more than one factor**.

For example, let's say we're predicting a house price.

- One predictor might be: square footage

- But what about: number of bedrooms? location? year built?

That's where **Multiple Linear Regression** comes in.

Multiple Linear Regression (MLR) generalizes the model to include **multiple independent variables**:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X_1 + \hat{\beta}_2 X_2 + \cdots + \hat{\beta}_p X_p \tag{87}$$

Where:

- Y: outcome.

- $X_1, X_2, \ldots, X_p$: predictors.

- β_0: intercept.

- β_j: effect of predictor j.

❓ Why use MLR?

Use Case	MLR Benefit
We want better predictions	More predictors, more information
We want to isolate effects	Estimate each variable's impact, **controlling for others**
We want realism	Real systems depend on **many factors**, not just one

4.6.1   Model Expansion

So far, we've worked with:

$$Y = \beta_0 + \beta_1 X + \varepsilon$$

This is a **simple linear regression (SLR)** with only **one predictor** (page 101).

We now expand this to:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon \tag{88}$$

This is **Multiple Linear Regression (MLR)** with p predictors.

✅ What's new here?

We now have **multiple input variables**, not just one X, but many contributions. Each gets its own coefficient β_j, representing its **independent contribution** to predicting Y.

📗 Intuition

Each β_j answers: "*if we increase X_j by 1 unit, how much does Y change, **holding all the other X's constant**?*". The last part is critical because, in MLR, we <mark>isolate the effect of each variable while controlling for the effects of the others</mark>.

For example, let's say we model student grades:

$$\hat{Y} = \beta_0 + \beta_1 \cdot \text{StudyTime} + \beta_2 \cdot \text{Sleep} + \beta_3 \cdot \text{Caffeine}$$

Then:

- β_1: how grades change **per hour of study**, keeping sleep and caffeine constant.

- β_3: how grades change **per mg of caffeine**, keeping study and sleep constant.

⚠️ Beware: Correlation Between Predictors

If predictors (e.g., Sleep and StudyTime) are **highly correlated**, it becomes harder to isolate the effect of one from the other. This is known as **multicollinearity**, and it can:

- ✖ Make coefficients unstable.

- ✖ Inflate standard errors.

- ✖ Confuse interpretation.

More formally, **Multicollinearity** occurs when two or more predictor variables in a multiple regression model are **highly correlated** with each other. This means they carry **overlapping information** about the outcome Y, making it hard for the model to distinguish their individual effects.

❷ **Why Multicollinearity is a Problem?** If X_1 and X_2 are strongly correlated, the model can't **tell who's responsible** for changes in Y. The estimated coefficients $\hat{\beta}_1$, $\hat{\beta}_2$ become **unstable**:

- They can change drastically with small changes in data.

- They may have **unexpected signs** (positive/negative).

- Their **standard errors** become large, then **t-tests become unreliable**.

For example, image this: $X_1 = \text{StudyTime}$, $X_2 = \text{TimeAtLibrary}$. If students who study more also spend more time at the library, then $X_1 \approx X_2$. So the regression model says: "*wait... both variables kind of say the same thing. Who gets the credit?*". The answer is that the model doesn't know, and the estimates become inaccurate.

❷ **How to detect Multicollinearity?**

Method	Description
Correlation matrix	Check if predictors are highly correlated
Variance Inflation Factor (VIF)	Measures how much a variable is explained by other predictors
Unstable coefficients	Coefficients flip sign or have huge standard errors

✅ **When is it *not* a problem?**

Multicollinearity **does not hurt prediction accuracy** (if all predictors are used). But it <u>does</u> hurt Interpretability, and Statistical Inference (t-test and p-values).

4.6.2   Matrix Formulation

❷ Why Use Matrix Form?

Matrix notation gives us:

- **Compact representation** of the model.

- A way to handle **any number of predictors** with one formula.

- Clean derivation of **OLS estimators**.

📋 Model in Matrix Form

We rewrite:
$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

As:
$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{89}$$

Where:

- $\mathbf{y}$: $n \times 1$ column vector of observed responses.

- $\mathbf{X}$: $n \times (p+1)$ **design matrix**, including a column of ones for the intercept.

- $\boldsymbol{\beta}$: $(p+1) \times 1$ vector of coefficients.

- $\boldsymbol{\varepsilon}$: $n \times 1$ vector of errors (residuals)

For example, if we have two predictors (plus intercept):

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix}$$

✅ OLS Solution Using Matrix Algebra

To estimate the coefficients that minimize residual sum of squares:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \tag{90}$$

This generalizes the simple regression formula to **any number of predictors**. It is the **Ordinary Least Squares estimators in matrix form**.

- $\mathbf{X}$: design matrix (rows are the data points, columns are the predictors +1).

- $\mathbf{X}^T$: transpose of $\mathbf{X}$.

- $\mathbf{X}^T \mathbf{X}$: matrix of sums of products, square and invertible if predictors are independent.

- $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$: inverse matrix, handles "division" in multidimensional space.

- $\mathbf{X}^T\mathbf{y}$: vector of dot products between predictors and output.

- $\hat{\beta}$: vector of estimated coefficients $\left[\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p\right]$.

📗 Predictions and Residuals

- **Fitted values**

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} \tag{91}$$

  Where:

  - $\hat{\mathbf{y}}$ is the vector of **fitted (predicted) values** for all our data points.
  - $\mathbf{X}$ is the **design matrix** with all our predictors and a column of ones (for the intercept).
  - $\hat{\beta}$ is the vector of **estimated coefficients**.

  This gives us all the predicted outcomes using the regression line we just fit.

- **Residuals**

$$\hat{\boldsymbol{\varepsilon}} = \mathbf{y} - \hat{\mathbf{y}} \tag{92}$$

  Where:

  - $\hat{\boldsymbol{\varepsilon}}$ is the vector of **residuals**, the errors the model made.
  - The residuals is the difference between:
    * $\mathbf{y}$ is the **true observed values**.
    * $\hat{\mathbf{y}}$ is the **predicted values** from your model. It is used for evaluating how well the model fits.

  In other words: "*how far off was our prediction from the actual result?*".

5   Model Selection and Regularization

5.1   Motivation & Problem Setup

✅ **The Goal.** We learned *how* to evaluate a model and which key factors to consider. Now, we want **to build** a model that is as simple as possible, but still accurate when predicting new data. In other words, to **find a linear model that is accurate**, **stable**, and **easy to interpret**. More precisely, we want to:

- ✔ **Make Good Predictions**. <mark>Not just on training data, but on new unseen data.</mark> This is why we care about generalization and not just high R^2 (coefficient of determination, page 110).

- ✔ **Avoid Overfitting**. Including too many predictors, our model fits noise. We want to choose only the useful variables.

- ✔ **Reduce Variance of Coefficients**. If variables are correlated, coefficients jump around a lot. We want stable estimates even if the data changes slightly.

- ✔ **Interpret the Model Easily**. Simpler models are easier to explain. Sparse models (with some $\beta_j = 0$) highlight the important variables.

🛠 **Setup: Linear Regression.** We have data:

- Y: outcome (like house price, disease risk, exam score)

- $X_1, X_2, \ldots, X_p$: features (like square footage, age, sleep hours)

We want to model:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

Using Ordinary Least Square (OLS, page 102), we estimate:

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T Y$$

It gives us the "best" coefficients to fit the data (minimize squared error).

⚠ **Problems.** However, this approach has some problems:

- **Too Many Variables?** The more variables we add, the higher R^2 goes. But adding variables blindly causes **overfitting**, and the model fits noise, not the true signal.

- **Multicollinearity** (page 127). If two variables $X_1 \approx X_2$, the OLS estimate becomes unstable. Interpretation of β_j becomes meaningless. The **variance** of the coefficients explodes:

$$\text{Var}\left(\hat{\beta}\right) = \sigma^2 \left(X^T X\right)^{-1}$$

If $X^T X$ is close to singular (not invertible), $\hat{\beta}$ becomes extremely variable!

- **High Variance, Low Bias**. OLS has **low bias** (accurate on average), but if predictors are correlated or too many, **variance** is high, then **poor generalization**.

5.2   Collinearity

5.2.1   Definition

Collinearity means that one predictor variable is **highly correlated with one or more** of the other predictors.

- The **Perfect Collinearity** is when one **variable is an exact linear combination of others**:

$$X_j = \alpha_0 + \sum_{k \neq j} \alpha_k X_k \tag{93}$$

- The **High Collinearity** (but not perfect), is when variables are almost linearly related (like $X_1 \approx X_2$).

⚠ Why is this a problem?

1. **OLS becomes unstable**: The matrix $X^T X$ becomes **ill-conditioned**[5] or **non-invertible**. The variance of $\hat{\beta}_j$ becomes **very large**.

2. **Interpretation breaks down**: It's unclear which variable is responsible for the effect on Y. We might see strange signs or inflated coefficients.

3. **Poor prediction**: Even if our training error is low, generalization on new data suffers.

❷ Impact on Variance of $\hat{\beta}$

Collinearity **doesn't hurt prediction** much (model can still fit Y well). But it makes our $\hat{\beta}_j$ estimates:

- **Unstable**: they swing wildly if we change data a little.

- **Hard to interpret**: the model is unsure whether to assign effect to X_1 or X_2.

It happens because the **variance of $\hat{\beta}$** becomes large (*effect*) when predictors are collinear (*cause*). This is **not just a side effect**, it **is** the mechanism by which collinearity causes damage.

The variance of $\hat{\beta}$ is:

$$\hat{\beta} \sim \mathcal{N}_p \left(\beta, \sigma^2 \left(X^T X \right)^{-1} \right)$$

---

[5]An **Ill-Conditioned matrix** is a matrix that has a high condition number, indicating that small changes in the input can cause large changes in the output. This makes the matrix sensitive to errors and can lead to significant inaccuracies in numerical computations.

This formula means that the **distribution** of our OLS coefficients $\hat{\beta}$ is a **multivariate normal** with:

- Mean = true coefficients β

- Covariance (variance) $= \sigma^2 (X^T X)^{-1}$

❓ **Okay, so what's the point of this variance?** The interesting part is $(X^T X)^{-1}$ because is the inverse of a **matrix that summarizes how our predictors relate to each other**:

- If predictors are **independent**, this matrix is *nice*, so inverse is stable and the **variance** is **small**.

- If predictors are **collinear**, this matrix is *close to singular*, so inverse is huge and the **variance explodes**.

Example 1: Analogy 1D

In 1D:
$$\mathrm{Var}(\hat{\beta}) = \frac{\sigma^2}{\sum x_i^2}$$

- If all x_i values are **spread out**, then the **denominator** is **large**, and the **variance** is **small**.

- If all x_i values are **clustered** (close together), then the **denominator** is **small**, and the **variance** is **large**.

In multivariable OLS that denominator becomes $X^T X$ and the variance becomes:
$$\mathrm{Var}(\hat{\beta}) = \sigma^2 \left(X^T X\right)^{-1}$$

☣ **Why Variance "explodes" with Collinearity?** If $X_1 \approx X_2$, then the matrix $X^T X$ becomes almost **non-invertible**. Its inverse $\left(X^T X\right)^{-1}$ contains **huge numbers**. That means that even a small change in the data can cause a large change in $\hat{\beta}$. This is why our model is "*wiggly*", it's jumping around because it's uncertain which variable to credit.
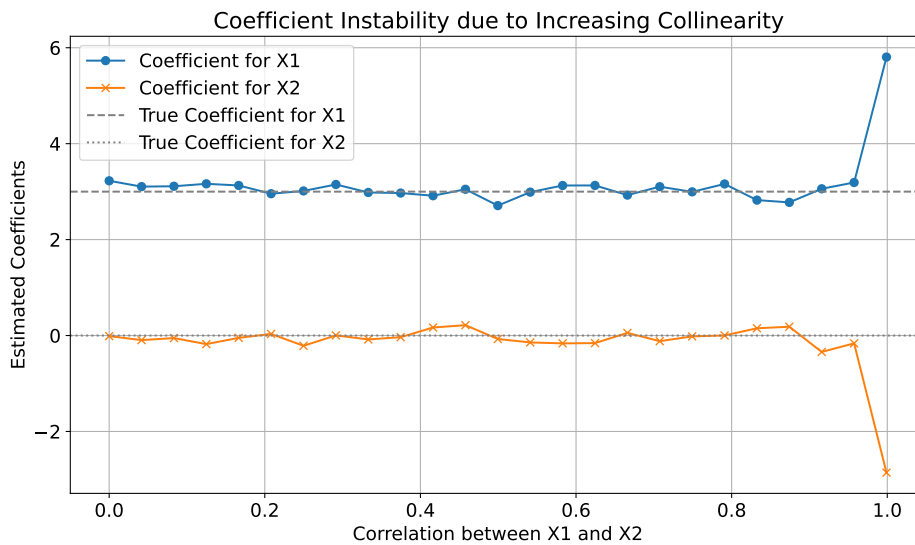
Figure 24: Coefficient Instability due to Increasing Collinearity.

- **X-Axis**: shows how strongly the two predictor variables X_1 and X_2 are related. We vary correlation from 0 (completely independent) to nearly 1 (almost identical).

- **Y-Axis**: these are the OLS estimates $\hat{\beta}_1$ and $\hat{\beta}_2$. The **true model** is:

$$Y = 3X_1 + \text{noise}$$

  So the "*truth*" is $\beta_1 = 3$, $\beta_2 = 0$.

- **Blue line** ($X1$): Estimated $\hat{\beta}_1$ (should be near 3).

- **Orange line** ($X2$): Estimated $\hat{\beta}_2$ (should be near 0).

- **Gray dashed line** (3): True value for β_1.

- **Gray dotted line** (0): True value for β_2.

❓ **What Happens as Correlation Increases?**

✅ Correlation near 0: **Low**. OLS correctly identifies X_1 as important, and $\hat{\beta}_1 \approx 3$, $\hat{\beta}_2 \approx 0$.

⚖️ Correlation: **Medium**. The model gets slightly confused: some effect is "shared" between both.

⚠️ Correlation near 1: **High**. The model becomes unstable: $\hat{\beta}_1$ and $\hat{\beta}_2$ swing wildly, but still together approximate 3.

This is the essence of **multicollinearity**. The model can't distinguish the true source of variation. It becomes **numerically unstable** even though prediction may still seem "okay".

5.2.2 Variance Inflation Factor (VIF)

Variance Inflation Factor (VIF) measures **how much the variance** of a regression coefficient is **inflated due to multicollinearity**. It quantifies the severity of multicollinearity in an OLS regression.

❓ How is VIF defined?

For a coefficient $\hat{\beta}_j$, the VIF is:

$$\text{VIF}_j = \frac{1}{1 - R_j^2} \tag{94}$$

Where R_j^2 is the **coefficient of determination** when **regressing X_j on all other predictors**.

Usually in regression, we do this:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

But for VIP, we switch things up. **Instead of using X_j to predict Y**, we do:

$$X_j = \gamma_0 + \gamma_1 X_1 + \gamma_2 X_2 + \cdots + \gamma_{j-1} X_{j-1} + \gamma_{j+1} X_{j+1} + \cdots + \gamma_p X_p + \eta$$

So we make X_j the dependent variable, and we use all other X_k (except X_j) as independent variables. Then we compute the R^2 of this regression.

- If $R_j^2 = 0$: X_j is independent, then no multicollinearity and $\text{VIF}_j = 1$.

- If $R_j^2 = 0.9$: 90% of X_j's variance is explained by others, then $\text{VIF}_j = \frac{1}{1 - 0.9} = 10$, danger!

VIF value	Interpretation
VIF $= 1$	✅ No correlation with other variables.
$1 < \text{VIF} < 5$	⚖️ Moderate correlation, usually okay.
VIF > 5	🔺 High correlation, potential multicollinearity.
VIF > 10	☠️ Very high correlation, serious problem.

Table 10: Interpretation of the VIF.

✖ Why it works?

The variance of the OLS estimate $\hat{\beta}_j$ is:

$$\text{Var}(\hat{\beta}_j) = \frac{\sigma^2}{(1 - R_j^2) \cdot S_{jj}}$$

Where:

- σ^2 is the noise variance.

- S_{jj} is the variance of X_j itself (scaled sum of squares).

- R_j^2 is how well X_j is explained by the other variables.

If R_j^2 is high ($R_j^2 \to 1$), then $1 - R_j^2$ is small (denominator $\to 0$). This is the **mechanism** behind multicollinearity, the more predictable X_j is from the others, the more **uncertain** our model is about how much X_j actually contributes to predicting Y.

So Variance Inflation Factor (VIF) is a **multiplier**, it tells us **how much larger the variance of $\hat{\beta}_j$ is due to collinearity**. For example, if $R_j^2 = 0.95$, then $\text{VIF}_j = \frac{1}{0.05} = 20$, and the variance of $\hat{\beta}_j$ is 20× worse than it would be with independent variables.
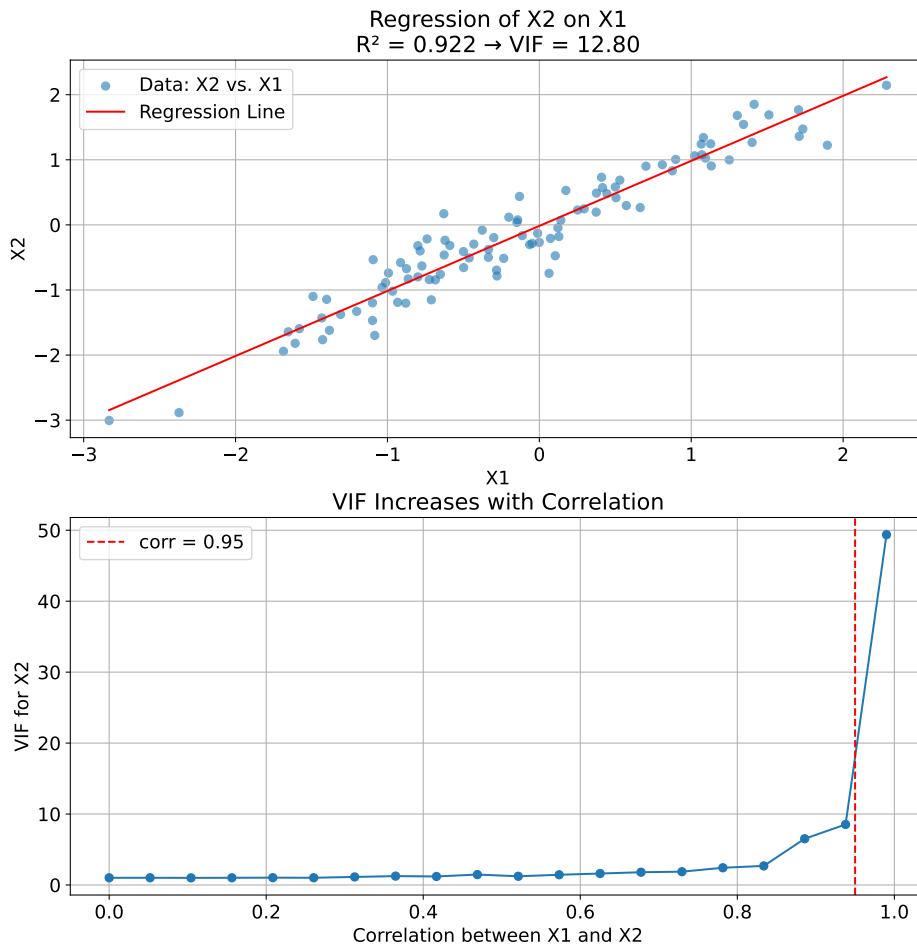
Figure 25: Top Plot: Regression Line. The **line fits the data well visually**, it looks like a strong relationship. Indeed, the $R^2 = 0.903$: 90% of the variability in X_2 is explained by X_1. But this is **not a good thing in this context**.

Bottom Plot: VIF Curve. The correlation of 0.95 (marked in red) leads to VIF ≈ 15. Even though the fit looks great, the **usefulness of X_2 in a regression model with X_1** is highly questionable.

A regression line might **look clean and convincing**, but if the predictor is **highly collinear**, our model is **mathematically unstable**. The **VIF reveals what the plot hides**: danger beneath the surface.

5.3   Handling Categorical Variables

Categorical Variables are **non-numeric variables** that represent categories or groups. For examples, ethnicity (Caucasian, Afroamerican, Middle-Eastern), degree program (CS, Math, Architecture). They are **qualitative**, not quantitative, so we can't just plug them into a linear model directly.

We often collect them directly from: questionnaire responses, database fields, CSV/Excel data with labeled columns.

Example 2: Categorical Variables

Our dataset is composed as follows:

Age	Gender	Ethnicity	Major
21	Male	Caucasian	CS
22	Female	Afroamerican	Architecture
23	Male	Middle-Eastern	Math

- Gender, Ethnicity, Major: categorical.

- Age: numeric.

❓ **Why should we include them in a model?**

Because they often explain **important, structured variation** in our outcome variable Y. For example, suppose we're modeling student GPA Y. The model with only numeric is:

$$Y = \beta_0 + \beta_1 \cdot \text{Hours Studied} + \varepsilon$$

But let's say **different majors** have different grading systems: CS has harder exams, and Architecture grades more leniently. Adding *Major* as a categorical variable improves the model:

$$Y = \beta_0 + \beta_1 \cdot \text{Hours} + \beta_2 \cdot \text{Major}_1 + \beta_3 \cdot \text{Major}_2 + \cdots + \varepsilon$$

❓ **How do we include them in Regression?**

We **convert each category into a dummy variable** (one-hot encoding). **One-Hot Encoding** is the standard method to convert categorical variables into numeric form so that they can be used in regression or machine learning models.

Definition 1: One-Hot Encoding

One-Hot Encoding transforms a categorical variable with K distinct categories into K binary (0 or 1) variables, **one for each category**.

Each binary variable indicates **whether that observation belongs to that category or not**.

Example 3: One-Hot Encoding

Suppose we have a categorical variable:

$$\text{Education} = [\text{“High School”}, \text{“Bachelor”}, \text{“Master”}, \text{“PhD”}]$$

We cannot directly plug this string into a regression. So, we use one-hot encoding:

Education	High School	Bachelor	Master	PhD
High School	1	0	0	0
Bachelor	0	1	0	0
Master	0	0	1	0
PhD	0	0	0	1

Each row has **exactly one 1**, all others are 0, hence the name “one-hot”.

But in Linear Regression, we **must drop one dummy column**. ❓ **Why drop a dummy column?** To avoid perfect multicollinearity (page 131), known as the *dummy variable trap*.

⚠ The Dummy Variable Trap

The **Dummy Variable Trap** happens when we include all categories of a categorical variable in a regression model. This causes **perfect multicollinearity**.

For example, let's say the variable Color has 3 categories:

$$\text{Color} \in \{\text{Red}, \text{Green}, \text{Blue}\}$$

We one-hot encode:

Observation	Red	Green	Blue
A	1	0	0
B	0	1	0
C	0	0	1
D	1	0	0

Now look:

$$\text{Red} + \text{Green} + \text{Blue} = 1 \quad \text{(for every row)}$$

So these columns are **linearly dependent**. This makes $X^T X$ **non-invertible** and the OLS fails. That's the **dummy variable trap**: we make our matrix non-invertible by including all dummies.

✅ **Solve the Dummy Variable Trap.** We must drop one dummy to make the system full-rank:

Observation	Green	Blue	(Red is reference)
A	0	0	
B	1	0	
C	0	1	
D	0	0	

Now the columns are **independent**, and regression works. ❓ **But what does this mean for the model?** When we drop "Red":

- The **intercept** β_0 is the predicted value for Red.

- β_{Green}: how much higher/lower Green is compared to Red.

- β_{Blue}: same, relative to Red.

So, **the dropped category becomes the reference group**, called **Reference Category**. All other group effects (dummy coefficients) will be interpreted relative to this group.

Question	Answer
What are categorical variables?	Text/label features like gender, city, or major.
Where do they come from?	Surveys, databases, user input, logs, real-world data.
Why include them in regression?	They capture **group-level effects** on the outcome.
How to include them?	Use dummy variables (one-hot encoding).
What is One-Hot Encoding?	Converts categorical variable into multiple 0/1 variables.
Why is One-Hot Encoding needed?	Regression models require numerical input.
Why drop one column?	To prevent multicollinearity (dummy variable trap).
How should it be chosen?	It depends on the reference group we are looking at.

Table 11: Summarize Categorical Variables.

❓ **Which column should we drop?**

We **don't randomly drop a column**, **nor** do we **base it on a strict mathematical heuristic**. Instead, the dropped column is ==chosen based on modeling intent== and sometimes default behavior of the software. For example, the StatsModels Python package usually drops the first level it encounters. However, the best practice is to decide which level makes the most sense based on our analysis.

Heuristic	When to use it
Most frequent category	Makes interpretation more stable and meaningful
Neutral or control condition	Good for experiments (e.g., placebo vs. treatment)
Lowest or baseline level	E.g., High School in Education, Intern in Job Type
Domain-specific standard	E.g., "Male" or "White" often used as baseline in social science

Table 12: "Heuristics" to choose the Reference Group.

Example 4: "Heuristics" to choose the Reference Group

Say we're studying income by education level:

- High School

- Bachelor

- Master

- PhD

If we want to know: "*How much more do people earn* **compared to High School** *graduates?*". Then we should **drop High School**. The coefficients will tell us how much Bachelor, Master, PhD differ from that baseline.

Finally, we **must drop a category** to avoid the Dummy Variable Trap. **Which category we drop** affects how we interpret the model, not the predictions, and **is up to us**.

▤ Interaction Terms with Dummies

Interaction Terms with Dummies let us model situations where the effect of a numeric variable (like hours studied) **depends on the category** (like gender or major).

Normally, we assume:
$$Y = \beta_0 + \beta_1 X_e + \varepsilon$$

Where X_e is a numeric variable (like experience, hours of study). But maybe the **effect of X_e depends on a categorical group** (e.g., gender). So we build a model like:

$$Y = \beta_0 + \beta_1 \cdot X_e + \beta_2 \cdot D + \beta_3 \cdot (X_e \cdot D) + \varepsilon$$

Where:

- $D = 1$ if the person is Male, 0 otherwise (e.g., Female is reference).

- $X_e \cdot D$ is the **interaction term**.

Example 5: Interaction Terms with Dummies

Say we believe:

$$Y = \text{Score} = \beta_0 + \beta_1 \cdot \text{StudyHours}$$

We assume everyone benefits equally from study hours. That's our baseline.

We would now like to upgrade our model to create more detailed results. So, we wonder: "*does **gender** affect score?*". So we add a dummy (categorical variable):

$$Y = \beta_0 + \beta_1 \cdot \text{StudyHours} + \beta_2 \cdot \text{Male}$$

Now we're saying:

- Everyone has the same slope (β_1). In a linear regression, the **Slope** is the **coefficient that multiplies a numeric predictor**. Here, β_1 is the slope and StudyHours is the numeric variable. In other words, it tells us how much Y changes when StudyHours increases by 1 unit (e.g., if $\beta_1 = 4.2$, then for every additional hour studied, the predicted exam score increases by 4.2 points).

- But males get a bump in baseline score ($+\beta_2$).

Then we wonder: "*maybe **study hours work differently** for males than females*". So we add:

$$\beta_3 \cdot (\text{StudyHours} \times \text{Male})$$

And now the model becomes:

$$Y = \beta_0 + \beta_1 \cdot \text{StudyHours} + \beta_2 \cdot \text{Male} + \beta_3 \cdot (\text{StudyHours} \cdot \text{Male})$$

❓ But what does this mean? Let's unfold it by case:

- Female (Male = 0):

$$Y = \beta_0 + \beta_1 \cdot \text{StudyHours}$$

- Male (Male = 1):

$$Y = (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \cdot \text{StudyHours}$$

  Here β_2 is the extra baseline score for males, and β_3 is the extra slope for males (*do they benefit more/less from studying?*)

❓ How do I know how to add a categorical variable to my regression model?

Let's say:

- X is a numeric predictor (e.g., study hours).

- C is a categorical variable with K levels: $c_1, c_2, \ldots, c_K$. In the final expanded formula, the categorical variable C itself **does not appear explicitly** because categorical variables are not numeric, so we can't use them directly in regression formulas. That's why we **replace** C with a set of **dummy variables**: $C \longrightarrow D_2, D_3, \ldots, D_K$. Each D_j is a 0/1 dummy variable.

- We choose one level (e.g., c_1) as the **reference group** (so to drop)

Then the model becomes:

$$Y = \beta_0 + \beta_1 X + \sum_{j=2}^{K} \gamma_j D_j + \sum_{j=2}^{K} \delta_j \left(X \cdot D_j \right) + \varepsilon \tag{95}$$

Where:

- β_0: intercept for the reference group.

- β_1: slope for the reference group.

- γ_j: intercept shift for group j.

- δ_j: slope shift for group j.

- $D_j = 1$ if the observation belongs to category c_j, 0 otherwise.

- $X \cdot D_j$ is the interaction between the numeric variable and the dummy for category c_j.

Example 6: Model with 4 categorical variables

Let's say:

- X = study hours.

- C = Major = {"CS", "Math", "Architecture", "Economics"}

- We choose CS as reference group.

We create:

- $D_{\text{Math}} = 1$ if major is Math, else 0.

- $D_{\text{Arch}} = 1$ if major is Architecture, else 0.

- $D_{\text{Econ}} = 1$ if major is Economics, else 0.

Then the model becomes:

$$\begin{aligned} Y = \quad & \beta_0 + \beta_1 X + \gamma_2 D_{\text{Math}} + \gamma_3 D_{\text{Arch}} + \gamma_4 D_{\text{Econ}} \\ & + \delta_2 (X \cdot D_{\text{Math}}) + \delta_3 (X \cdot D_{\text{Arch}}) + \delta_4 (X \cdot D_{\text{Econ}}) + \varepsilon \end{aligned}$$

5.4   Variable Selection

5.4.1   Definition

Variable Selection (or **Feature Selection**) is the process of choosing **which predictors (X's)** to include in our model.

❓ Why do we need it?

In real-world data, we often have **many predictors** and some are:

- Irrelevant

- Redundant

- Weakly related to Y

- Strongly correlated with others (collinearity)

Then including all variables can:

✖ Inflate variance (*remember multicollinearity?*).

✖ Make the model **overfit** to noise.

✖ Reduce interpretability.

✖ Add unnecessary complexity.

So we want to **select a subset of variables** such that:

✔ Gives good predictive performance.

✔ Keeps the model simple and interpretable.

🗏 Model Selection Problem

Given a full set of p predictors:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon$$

We want to find a **smaller model** (called *Model Selection Problem*):

$$Y = \beta_0 + \sum_{j \in S} \beta_j X_j + \varepsilon$$

Where S is a subset of the predictors (i.e., the best combination). The **Model Selection Problem** is the challenge of choosing **the best model** from a **collection of candidate models**. We want to **fine the one** that best explains or predicts the data without overfitting.

Formally, given a set of candidate models $\mathcal{M}_1, \ldots, \mathcal{M}_K$, find the model $\mathcal{M}_j$ that **minimizes a chosen criterion**, such as: Validation error, AIC / BIC, Cross-validation error, Mallows' C_p, Adjusted R^2.

❓ **Why is this so important? We can do it by hand, right?** Say you have $p = 10$ predictors: $X_1, X_2, \ldots, X_{10}$. There are $2^{10} = 1024$ possible subsets of predictors! Each subset defines a **different model**:

$$Y = \beta_0 + \sum_{j \in S} \beta_j X_j + \varepsilon$$

Where $S \subset \{1, 2, \ldots, 10\}$. So the **model selection problem** becomes: *which subset S of predictors gives the best model?*

❓ **What makes a "Good" subset?** That depends on our goal. We might want the model that:

- Has the **lowest test error** (prediction accuracy).

- Uses the **fewest predictors** (simplicity).

- Balances both (e.g., via AIC, BIC, cross-validation).

⚖️ **Tradeoff: Bias vs. Variance.** Choosing too many variables:

✔️ Low bias (flexible).

✖️ **High variance**, then overfits noise.

But, choosing too few:

✖️ High bias (underfits).

✔️ Low variance.

So model selection is about finding the **right tradeoff**.

⚠️ **How hard is it to find a *good subset*?** Short answer: **hard**. Because the number of models grows **exponentially** with the number of predictors. Also, test error isn't known in advance, we must estimate it. And finally, some variables are **marginally useless** but **jointly helpful**, and vice versa. So it's not so free.

📋 **Common Strategies**

Method	Idea
Best Subset Selection (page 160)	Try all possible combinations and choose the best.
Forward Stepwise (page 163)	Start with none, add one at a time.
Backward Stepwise (page 165)	Start with all, remove one at a time.
Ridge Regression (page 169)	Shrink coefficients, never removes variables (L2 norm).
Lasso (page 172)	Shrink and set some coefficients to 0 (L1 norm).
Elastic Net (page 175)	Combines Ridge and Lasso.
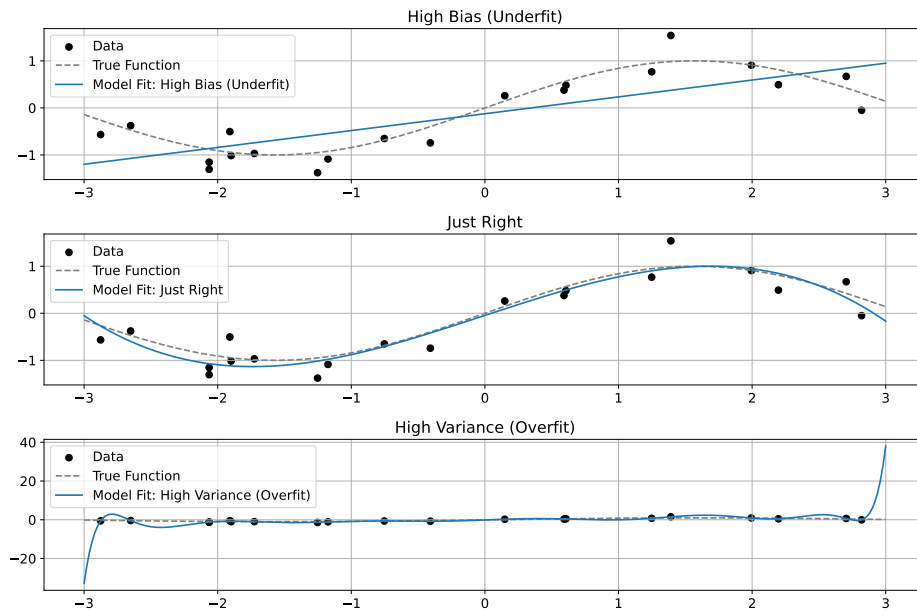
Figure 26: **Summary of the bias-variance tradeoff**.

- **High Bias (Underfit)** (top). Linear model (degree 1) too simple to capture the true pattern (a sine wave).

  - ✘ **High bias**: Misses key features.
  - ✔ **Low variance**: Prediction is stable.

- **Just Right** (middle). Polynomial of degree 3 that captures the main structure without chasing noise. Good generalization: **low bias**, **low variance**.

- **High Variance (Overfit)** (bottom). Polynomial of degree 15 that fits noise in the training data perfectly.

  - ✘ **High variance**: Prediction is unstable.
  - ✘ **Too low bias**: Too flexible, will perform poorly on new data.

❓ **Why does the true function appear as a line rather than a sine wave?** Because the overfit model (a high-degree polynomial) produces such large, wiggly oscillations that it stretches the vertical axis to fit them. As a result:

- The **true function**, which is a smooth sine wave, has relatively small amplitude (between -1 and 1).
- The **fitted curve** might swing up to $+10$ or down to -10 or more.
- This **resizes the entire plot**, making the sine wave appear almost flat, like a straight line.

5.4.2   Model Assessment and Selection Criteria

In the process of building predictive models, selecting the right model is as crucial as fitting it. While more complex models can often fit training data more accurately, they are also more prone to **overfitting**, capturing noise rather than underlying patterns. Conversely, overly simple models may **underfit**, failing to capture important structure in the data.

This chapter focuses on techniques used to **evaluate and compare models**, with the goal of identifying the one that best balances **predictive accuracy** and **model complexity**. These methods aim to estimate a model's **generalization error** (its expected performance on unseen data) using either **data resampling techniques** or **penalized evaluation metrics**.

We will present both **empirical methods**, such as the **Validation Set** and **Cross Validation**, and **analytical criteria**, such as **Adjusted R^2**, **AIC**, **BIC**, and **Mallows' C_p**. Each method offers a different trade-off between **bias**, **variance**, and **computational cost**, and is suitable for different settings depending on data size and modeling goals.

📗 Some terms to know before delving in

- The **Test Error** (also called **Generalization Error**) is the **expected error the model makes when predicting new, unseen data**. It reflects how well the model performs **in practice**, not just on the training data.

  Formally, if the true relationship is:

  $$Y = f(X) + \varepsilon$$

  And the model is $\hat{f}(X)$, then the **test error** is usually measured as:

  $$\mathbb{E}_{(X,Y)} \left[(Y - \hat{f}(X))^2 \right] \tag{96}$$

  Where:

  - $\mathbb{E}_{(X,Y)} [\dots]$: The **expected value** over the joint distribution of input X and output Y. It is basically the **mean** or **average** of a random quantity, in other words, take the average of $(Y - \hat{f}(X))^2$ **over all possible values** of X and Y.

    In practice, we **draw many new data points**, (X, Y), from the real world, not the training data, and compute the squared error for each one. Then, we take the **average** of all those errors.
  - Y: The **true output** of a new, unseen observation.
  - $\hat{f}(X)$: Our **model's prediction** for the input X.
  - $(Y - \hat{f}(X))^2$: The **squared error** between the true output and our model's prediction.

  This expectation is over **new data points**, not the ones used for training.

- A **Loss Function** quantifies how bad a model's prediction is compared to the true outcome. It measures the "cost" of an incorrect prediction.

  For example, if our model predicts:

    - True value: $y = 5$.
    - Prediction: $\hat{y} = 3$.

  Then the **loss** might be:

    - $(5 - 3)^2 = 4$ (squared error)
    - $|5 - 3| = 2$ (absolute error)

  The smaller the loss, the better the model's prediction.

5.4.2.1 Validation Set

The **Validation Set method** is a way to estimate a model's **test error** by splitting our data into:

- A **training set** (used to fit the model);

- And a **validation set** (to test how well the model performs on unseen data).

We then compute the **prediction error** (e.g., mean squared error) on the validation set. This serves as an **estimate of the test error**, i.e., how the model would perform in the real world on new data.

However, we do not calculate the true test error, we <mark>approximate it</mark>. The true test error is defined on future, unknown data. Since we don't have that, we simulate the situation by holding out part of our current data (the validation set) as a stand-in for future data.

⚒ How it works

1. **Split the data**. We divide the full dataset into two parts:

   - **Training set** (e.g., 60-80% of data): used to fit the model.
   - **Validation set** (e.g., the remaining 20-40%): used to estimate test error.

   This split should be **random** to avoid bias.

2. **Fit the model**. Using the **training set**, fit our model, for example, linear regression with a certain set of predictors or a decision tree with chosen parameters.

3. **Make predictions on the validation set**. Use the fitted model to predict the **response values** in the **validation set** (not used in training!).

4. **Compute the validation error**. Compare the predicted and actual values using a loss function (see page 147), like:

$$\mathrm{MSE_{val}} = \frac{1}{n_{\mathrm{val}}} \sum_{i \in \mathrm{validation\ set}} (y_i - \hat{f}(x_i))^2$$

   This is our **estimate of the test error**. We use the **Mean Squared Error (MSE)** in the validation set because:

   - It's **easy to compute and differentiate** for optimization reasons.
   - It heavily **penalizes large errors**, which can help detect unstable models.
   - It has **good theoretical properties** (like being the maximum likelihood estimator under Gaussian noise).

   MSE is commonly used for regression because it's mathematically convenient and sensitive to large errors.

✅ Purpose

The goal is to **simulate performance on unseen data**, using the validation set as a stand-in for future observations. We use this error to **choose the best model**.

⚠️ Limitations

- **High variance**: the result can depend heavily on which data points end up in the validation set.

- **Reduced training size**: we're not using the full data to train, which may affect model quality.

That's why more stable approaches like *cross-validation* are often preferred.

5.4.2.2   Cross-Validation (CV)

We will present a brief overview of the cross-validation method here, as it will be explained in more detail in future sections.

Cross-Validation (CV) is a technique to estimate a model's test error **more reliably** than the simple validation set method. Instead of holding out just one subset of the data, CV **rotates** the validation role across **multiple subsets**. This reduces variability and makes better use of the data.

☰ Types of Cross-Validation

- **k-Fold CV**: Split into k parts, train on $k-1$, validate on the remaining part. Repeat k times.

- **Leave-One-Out CV (LOOCV)**: Extreme case of k-fold where $k = n$; each fold has one data point.

- **Repeated k-Fold CV**: Repeat k-fold CV multiple times with different splits, then average results.

- **Stratified k-Fold**: Like k-fold but ensures each fold has a balanced class distribution (important in classification).

- **Monte Carlo CV** / **Shuffle-split**: Randomly split the data into training and test sets multiple times.

▤ Most common form: k-Fold Cross-Validation

1. **Split the data** into k roughly equal-sized **folds** (groups).

2. For each fold $i = 1, \ldots, k$:
   - Use all the data **except** fold i to **train** the model.
   - Use fold i as the **validation set**.
   - Compute the prediction error on fold i.

3. **Average** the k errors to get the **cross-validation estimate** of test error.

Example 7: 5-Fold CV

We split the dataset into 5 parts ($F1$ to $F5$). We run 5 iterations:

- Train on $F2 + F3 + F4 + F5$, test on $F1$.

- Train on $F1 + F3 + F4 + F5$, test on $F2$.

- And so on.

We average the 5 validation errors, and the result is the Cross-Validation error estimate.

❓ Why not use just one validation set?

Because the estimate would depend heavily on which data are held out. CV reduces this **variance** by **reusing all the data** for both training and validation (just not at the same time).

📗 Special k-Fold CV cases

Type	Description
Leave-One-Out CV	$k = n$: each fold is one single observation.
Repeated k-Fold CV	Repeat k-fold CV multiple times with different splits, then average.
Stratified k-Fold	Ensure each fold has balanced classes (used in classification).

Feature	k-Fold CV
Estimates test error?	✅ Yes
Uses all data for training?	✅ Eventually, each point is used $k - 1$ times
Reduces variance?	✅ Compared to single validation split
More computation?	⚠️ Yes, model is fit k times!

Table 13: Summary of k-Fold Cross-Validation.

5.4.2.3   Adjusted R^2

Adjusted R^2 is a modification of the regular R^2 (coefficient of determination, page 110) that adjusts for the **number of predictors** in the model. It helps prevent choosing an overly complex model just because it has more variable.

⚠ Problem with regular R^2

Remark. The coefficient of determination is defined as follows:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

It tells us what **fraction of the variance in** Y is explained by the model. For example, if $R^2 = 0.80$, that means 80% of the variance in the output Y is explained by the predictors. This sounds good! But there's a **catch**.

R^2 **never goes down** when we add more predictors, even **if those predictors are useless**! Why? Because:

- Every time we add a variable, our model becomes more **flexible**.

- That means it can **fit the training data better**, even if the new variable is **pure noise**.

- So **RSS goes down**, and R^2 goes **up**.

<u>But</u> a better fit on training data **does not mean** better prediction on new data! This is called overfitting (remember page 83?), the model learns noise instead of the signal.

👎 We are still not convinced. Let's make an example. Let's say we have:

- Model A with 2 predictors, and $R^2 = 0.82$.

- Model B with 10 predictors, and $R^2 = 0.88$.

Looks like Model B is better, right? Not necessarily, that is the catch! If the 8 extra predictors are **just random noise**, Model B might look good on training data (amazing), perform **worse** on test data (not so good). So regular R^2 tricks us into favoring **more complex models** even when they **don't generalize well**.
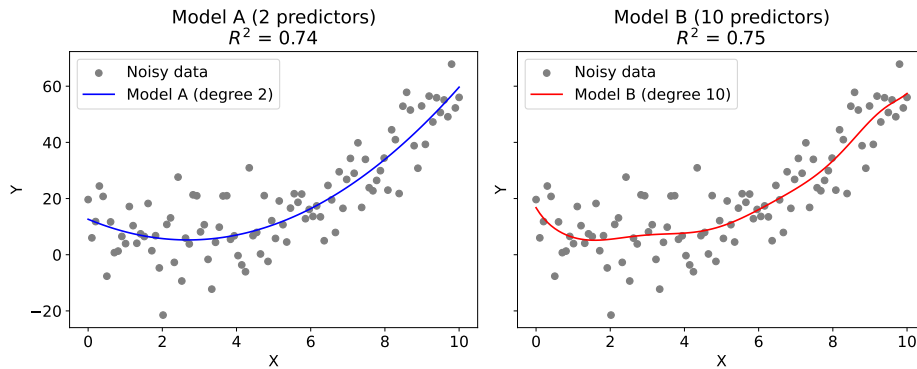
Figure 27: Two models fitted to the same noisy data. The Model A (left) is a quadratic model (only 2 predictors). It fits the general shape of the data well without overreacting to noise. The Model B (right) is a 10th-degree polynomial (10 predictors). It fits the training data very closely, so the R^2 is **higher**, but it's clearly **overfitting**, capturing noise instead of underlying pattern.

✅ Solution: Adjusted R^2

The Adjusted R^2 modification allows us to **penalize** the model for adding variables that **don't improve the fit enough**.

- ✅ If a new variable helps, Adjusted R^2 goes **up**.

- ❌ If a new variable doesn't helps, Adjusted R^2 goes **down**.

$$\text{Adjusted } R^2 = 1 - \frac{\frac{\text{RSS}}{(n-d-1)}}{\frac{\text{TSS}}{(n-1)}} = 1 - \left(\frac{\left(1 - R^2\right)(n-1)}{n - p - 1} \right) \tag{97}$$

Where:

- RSS: Residual Sum of Squares (the same value used in the normal R^2 formula).

- TSS: Total Sum of Squares (the same value used in the normal R^2 formula).

- n: number of **observations**.

- d: number of **predictors used**.

So now, when comparing models with different numbers of variables, we prefer the model with the **highest Adjusted R^2**.

⚠️ Adjusted R^2 is not immune to overfitting; it can be fooled

Adjusted R^2 **penalizes** adding too many variables. It makes overfitting **less likely**. It is **more reliable than** R^2 when comparing models with different numbers of predictors. **So yes**, it helps **reduce the risk** of overfitting, but **it does not eliminate** the risk.

Adjusted R^2 still uses **only training data**. So if a very complex model fits the training data **very well**, even by learning the **noise**, the RSS can drop **a lot**, and the penalty might not be enough to catch it. This is why Adjusted R^2 **helps control overfitting**, **but it can still be fooled**.

In general, it's much safer than R^2, because R^2 **always increases** when we add variables, instead of **adjusted R^2 increasing only** when a variable **really helps**. But that doesn't mean it **always chooses the best model**. It just gives us a **better chance** of avoiding overfitting, <u>**not a guarantee**</u>.

❷ **So how should we use this metric?** Use Adjusted R^2 as a guide, but not as a judge. Adjusted R^2 is a useful guide for comparing models of different complexity, but it should not be used as the sole decision criterion. It penalizes model complexity and helps detect overfitting, but since it is still based on training data, it can be misled when a more complex model fits noise. For more reliable model selection, especially when comparing models with similar adjusted R^2, it's essential to combine it with other tools such as cross-validation, residual analysis, or information criteria (like AIC/BIC).

Example 8: Analogy

Imagine we're packing a **toolbox** (our model) to fix problems in different houses (datasets).

- R^2 is like judging how many problems we *can* fix, the more tools we add, the more it praises us. Even if we throw in **wrenches we'll never use**, it says "Great! We're ready for anything!".

- Adjusted R^2 is like a smart supervisor who checks: "Okay, our toolbox is more effective, **but is it worth the extra weight?**". If our added tools actually help (reduce error), it says "Good job.", but if we just added unnecessary stuff, it **lowers our score**.

Adjusted R^2 rewards **useful tools**, but penalizes us if our box is bloated and heavy **for no reason**. But here's the <u>limitation</u>:

- This supervisor **never leaves the garage** (training set).

- They don't know if we'll actually be **faster or clumsier** when fixing real houses (test data).

For that, we need **cross-validation**, like sending us on test jobs to see how well we perform.

Adjusted R^2 is like a smart intern: they try to carry only the essential tools for the job, avoiding extra clutter. They're efficient, careful, and don't overpack. But, they've never worked outside the office. They've only practiced inside the training environment. So while they make good guesses about what's useful, they might still carry tools that won't help (overfitting), or miss tools that are actually needed.

❓ **So what are R^2 and Adjusted R^2 really useful for?**

1. **Descriptive insight**. They tell us, in simple terms:

   - How much of the variance in the response Y is explained by our model.
   - Whether adding variables seems to improve our model's fit (Adjusted R^2).

   In exploratory analysis[6], they're **very helpful** to:

   - Compare models **quickly**.
   - Communicate results.
   - Spot **obvious** **underfitting**.

2. **Model comparison when CV is expensive**. In large models or datasets, cross-validation can be:

   - ✖ Too slow.
   - ✖ Too expensive.

   Adjusted R^2, AIC, and BIC offer **quick heuristics** for model selection without resampling.

3. **Linear models with nested predictors**. In linear regression, where models are **nested** (Model A $\subset$ Model B, A subset of B), adjusted R^2 can effectively:

   - ✔ Tell whether adding variables helps.
   - ✔ Give a quick "cost-benefit" of complexity vs fit.

Metric	Useful for...	Limitation
R^2	🔰 Quick summary of fit	🔻 Always increases with complexity
Adjusted R^2	🔰 Penalized model comparison	❗ Still based on training data only
Cross-Validation	🔒 Realistic test error estimate	⏳ Slower, more compute-intensive

In summary, use R^2 and Adjusted R^2 for **quick insight**. Use cross-validation when **accuracy** and **generalization are important**.

---

[6]**Exploratory Data Analysis (EDA)** is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. [3]

5.4.2.4   AIC and BIC

Akaike Information Criterion (AIC), as named by its inventor Hirotugu Akaike, and **Bayesian Information Criterion (BIC)**, as named by its inventor Gideon Schwarz [7], are **model selection criteria** that balance two goals:

1. **How well the model fits the data** (like low RSS, page 102, or high likelihood).

2. **How simple the model is** (fewer predictors).

In simple terms, they answer this question: "*among many models, which one likely generalizes best to new data?*".

√× Formulas

Both are based on the **log-likelihood** of the mode, how well it fits, and then **penalize** it based on the number of parameters. So, they both start with the **log-likelihood** of the model (i.e., how likely is the data under this model), and then **penalize** it for how many parameters the model uses.

For a linear model with Gaussian errors:

$$\text{AIC} = n \cdot \log\left(\frac{\text{RSS}}{n}\right) + \underbrace{2d}_{\text{penalty}} \tag{98}$$

$$\text{BIC} = n \cdot \log\left(\frac{\text{RSS}}{n}\right) + \underbrace{d \cdot \log(n)}_{\text{penalty}} \tag{99}$$

Where:

- n: number of observations.

- d: number of predictors (model complexity).

- RSS: residual sum of squares (fit quality, see page 102).

- ✅ Both **reward** good fit (**low RSS**).

- ❌ Both **penalize** large d (**complexity**, too many predictors).

⚖️ What's the difference?

Criterion	Penalty for complexity	Tends to favor...
AIC	$2d$	Slightly more complex models
BIC	$d \cdot \log(n)$	Simpler models, stronger penalty

Table 14: A simple comparison between AIC and BIC.

	AIC	BIC
Penalty strength	Mild: $2d$	Stronger: $d \cdot \log(n)$
Tends to choose	Slightly more complex models	Simpler models
Best for	Prediction (lower test error)	Identifying true model (if exists)

Table 15: Interpretation of AIC and BIC.

So:

- **AIC** is better when we're focused on **prediction**.

- **BIC** is better when we're trying to **find the true model** (especially when n is large).

BIC is **more stringent** than AIC because its penalty is stronger, especially as n increases.

Situation	AIC says...	BIC says...
Slightly better fit, more parameters	✅ "Okay, that's fine"	❌ "Nope, not worth it."
Big dataset with noisy improvements	👍 "Still acceptable"	⚠️ "We're overfitting"

❓ How to use them?

1. Compute AIC and/or BIC for each candidate model.

2. **Choose the model with the lowest value**.

They are **not interpretable on their own**, they're only useful **in comparison** between models.

Example 9: What AIC and BIC do

Imagine we have 10 models. AIC and BIC will:

- Look at how well each model fits the data (small RSS = good).

- Subtract points if the model is **too complex**.

- Pick the model with the **lowest score**.

They answer: "*Which model fits well, **without overdoing it?***".

⚠ How do AIC and BIC relate to overfitting?

Overfitting occurs when a model is too complex and fits the noise instead of the signal. AIC and BIC can **prevent overfitting** because they **penalize unnecessary complexity**. However, they do so in different ways and with different levels of strictness, as we have seen above.

🛡 **BIC** is **more resistant to overfitting** than AIC.

❗ **AIC** still helps, better than Adjusted R^2, but may **accept a few extra variables** if they improve training fit slightly.

❓ If BIC is more robust, why can't we always use it instead of AIC?

Because **avoiding overfitting isn't the only goal**. Sometimes, **being too strict** can cause the opposite problem: **underfitting**.

BIC is **very conservative**. It heavily penalizes additional predictors, and often selects **very simple models**, especially as n (sample size) grows.

✅ This is good when our goal is to **identify the true underlying model**, such as in scientific discovery.

❌ But BIC can **miss useful predictors** that would improve **predictive accuracy**, especially:

- When the true model is complex.
- When some variables are only **mildly helpful**.
- When sample size is small (so penalty is not balanced by enough data).

Method	Immune?	Why/Why not
R^2	❌ ❌	Always increases with more variables, pure training fit.
Adjusted R^2	❌	Penalizes complexity, but still based on training data.
AIC	⚖	Penalizes complexity, but might allow slight overfit for predictive gain.
BIC	✅ (almost)	Strong penalty, favors simpler models, but still uses training data.
Cross-Validation	✅ (best in practice)	Tests the model on unseen data, closest thing to a real-world check.
Real test set	✅ ✅ (gold standard)	Directly measures generalization, but only if we have one!

Table 16: No method is 100% immune to overfitting, but some are much **more resistant**. This table shows the hierarchy of **resilience to overfitting**.

5.4.2.5   Mallows' C_p

Mallows' C_p is a metric designed to estimate **how much error** our model will make on **new data**, while **penalizing complexity**, just like AIC and BIC. It was developed specifically for **linear models** and works especially well in **Best Subset Selection** (page 160).

√x̄ Formula

For a linear regression model with d predictors and residual sum of squares RSS_d, Mallows' C_p is:

$$C_p = \frac{1}{\hat{\sigma}^2}\left(\text{RSS}_d + 2d \cdot \hat{\sigma}^2\right) \tag{100}$$

Where:

- RSS_d: Residual sum of squares for the model with d predictors.

- $\hat{\sigma}^2$: Estimate of the **irreducible error** (often from the full model).

We can use a **simplified version** when working with **linear regression models** or when we want a **quick comparison of different subset models**:

$$C_p = \frac{\text{RSS}_d}{\hat{\sigma}^2} - (n - 2d)$$

But be careful because this formula **depends on a good estimate of $\hat{\sigma}^2$**. If $\hat{\sigma}^2$ is **underestimated**, C_p will **favor too complex models** (under-penalize complexity). If overestimated, it might underfit. So, use the simplified formula when doing linear regression subset selection, and when we have a good estimate of the model's noise variance from the full model.

However, like AIC, it tries to **balance fit and complexity**. If our model is good (low bias, low variance), then $C_p \approx d$ (the number of predictors).

- If $C_p > d$, our model is likely **missing important variables**.

- If $C_p < d$, our model may be **overfitting**.

❷ How do we use it?

Compute C_p for each candidate model (especially in subset selection). Then, **choose the model where C_p is smallest**, and ideally **close to d**.

5.4.3   Best Subset Selection

The **Best Subset Selection** is a method to find the combination of predictors that produces the best-performing linear regression model. Among all possible subsets of predictors, we **search for the one that minimizes prediction error** (or maximizes Adjusted R^2, AIC, BIC, etc.).

✖ How it works

Suppose we have p predictors: $X_1, X_2, \ldots, X_p$:

1. For each $k = 0, 1, \ldots, p$:

   - Consider **all possible models** that contain **exactly k predictors**.
   - Fit all $\binom{p}{k}$[7] combinations of those predictors. Where p is the total number of available predictors, and k is the number of predictors we want to include in the model. In other words, is the number of different models we can build using **exactly k** of the p predictors.

2. Among all models of size k, choose the one with the **lowest training RSS**[8] (or highest adjusted R^2). The *lowest training RSS* is achieved by the model with all p predictors, and it is:

$$\min_{k=1,\ldots,p} \left(\min_{\text{model with } k \text{ predictors}} \text{RSS} \right) = \text{RSS of full model}$$

3. Compare the **best models of each size k** using a criterion: Validation Set (page 148), Cross-Validation (page 150), Adjusted R^2 (page 152), AIC / BIC (page 156), Mallows' C_p (page 159).

Step	Action
1	Try all possible subsets of predictors.
2	Find the best model for each number of predictors.
3	Choose the best overall model based on a criterion.

❷ When to use it

- When p is small (typically < 20).

- When interpretability matters.

- When we want to explore all possibilities.

---

[7]**Binomial Coefficient** gives the number of ways to choose k elements from a set of p elements **without regard to order**.

[8]RSS refers to the Residual Sum of Squares (page 102).

✅ Advantages

- ✔ **Exhaustive**: finds the optimal model **for each size**.

- ✔ Easy to **understand and explain**.

- ✔ Produces interpretable models.

❌ Disadvantages

- ✖ Computationally **expensive**: 2^p models to fit (e.g., 1,024 if $p = 10$, 1 million if $p = 20$).

- ✖ Not feasible for **large** p.

Example 10: Best Subset Selection

We want to predict exam score using these predictors:

- $X1$: study hours.

- $X2$: number of coffee cups.

- $X3$: hours of sleep.

- $X4$: prior GPA.

So we have $p = 4$ predictors. Our *goal* is to find **which combination** of these predictors gives the **best linear model** for predicting score. Not necessarily all 4, maybe only $X1$ and $X4$ are enough.

❓ **What does best subset do?** It tries **every possible combination** of predictors, and compares the models. Let's list them:

1. Try 1-variable models:

    - Model A: $Y = \beta_0 + \beta_1 X_1$
    - Model B: $Y = \beta_0 + \beta_2 X_2$
    - Model C: $Y = \beta_0 + \beta_3 X_3$
    - Model D: $Y = \beta_0 + \beta_4 X_4$

    For each model fit it on training data, then compute a score that tells how good it is (AIC / BIC, etc.). Finally, among these 4 one-variable models, we pick the best one.

2. Try 2-variable models:

    - Model A: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$
    - Model B: $Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3$
    - Model C: $Y = \beta_0 + \beta_1 X_1 + \beta_4 X_4$
    - Model D: $Y = \beta_0 + \beta_2 X_2 + \beta_3 X_3$

- Model E: $Y = \beta_0 + \beta_2 X_2 + \beta_4 X_4$
- Model F: $Y = \beta_0 + \beta_3 X_3 + \beta_4 X_4$

And we pick the best one.

3. Try 3-variable models:

- Model A: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$
- Model B: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_4 X_4$
- Model C: $Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \beta_4 X_4$
- Model D: $Y = \beta_0 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$

Then the best one.

4. Try all 4 predictors (only one): $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$

With 4 predictors, there are $2^4 = 16$ possible subsets, then 16 models to try. At the end, we **compare the best model** of each size, and then pick the one that gives the **lowest estimated test error**.

5.4.4   Forward Stepwise

The goal of **Forward Stepwise Selection** is to find a parsimonious model that **balances predictive power** with **simplicity**, <mark>without evaluating</mark> **all possible** combinations (as in best subset selection, page 160, which is computationally expensive).

The key idea is to **start with no predictors**. At each step, **add the predictor** that improves the model the most, **given the current set of predictors**. Continue until:

- A maximum number of predictors is reached;

- No significant improvement occurs;

- A criterion (like BIC or cross-validation error) suggests stopping.

🛠 How it works

Assume we have 5 predictors: X_1, X_2, X_3, X_4, X_5.

1. Start with the **null model**: $\hat{Y} = \bar{Y}$. It is the mean of the response variable Y, that is:
$$\bar{Y} = \frac{1}{n} \sum_{i=1}^{n} Y_i$$

2. Fit **5 models**, each with one predictor:
    - Model 1: $Y = \beta_0 + \beta_1 X_1$
    - Model 2: $Y = \beta_0 + \beta_2 X_2$
    - Model 3: $Y = \beta_0 + \beta_3 X_3$
    - Model 4: $Y = \beta_0 + \beta_4 X_4$
    - Model 5: $Y = \beta_0 + \beta_5 X_5$

    Choose the one with **lowest RSS** or **best score**. For example, we suppose it's X_3.

3. Now fit **4 models**, each adding one of the remaining variables to X_3:
    - Model 1: $Y = \beta_0 + \beta_3 X_3 + \beta_1 X_1$
    - Model 2: $Y = \beta_0 + \beta_3 X_3 + \beta_2 X_2$
    - Model 3: $Y = \beta_0 + \beta_3 X_3 + \beta_4 X_4$
    - Model 4: $Y = \beta_0 + \beta_3 X_3 + \beta_5 X_5$

    Choose the best again. For example, we suppose it's X_4. Now the model is: $Y = \beta_0 + \beta_3 X_3 + \beta_4 X_4$.

We repeat until a stopping rule is met (e.g. validation error increases).

✅ Advantages

- ⏱ **Much faster** than best subset selection (linear vs exponential).

- Often yields very similar results.

- Can be combined with information criteria (AIC/BIC), validation, etc.

❌ Disadvantages

- **Greedy** method[9]: once a variable is added, it **stays**, even if a better subset is possible without it.

- Might **miss the best model** if combinations are more powerful than incremental gains.

---

[9]A Greedy method is an algorithm that at each step, makes the best immediate choice without considering the long-term consequences.

5.4.5   Backward Stepwise

The key idea of **Backward Stepwise Selection** is to start with **all predictors** in the model. At each step, **remove** the **least useful** predictor, the one whose removal **improves** (or least worsens) the model based on a criterion like RSS, AIC, BIC, etc. Conceptually, Backward Stepwise Selection is the inverse of Forward Stepwise Selection. However, even though they move in opposite directions, they **don't necessarily reach the same model**. Since they are greedy algorithms, different paths can potentially lead to different final models.

It is **used** when the **number of predictors** (p) is **not too large**, or when we want a simpler model but **don't want to try all possible combinations** (as in best subset selection).

🛠 **How it works**

Assume we have 5 predictors: X_1, X_2, X_3, X_4, X_5.

1. Start with the **full model**:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_5 X_5 + \varepsilon$$

2. Fit **5 reduced models**, each omitting one predictor:

   - $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4$
   - $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_5 X_5$
   - $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_4 X_4 + \beta_5 X_5$
   - $Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5$
   - $Y = \beta_0 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5$

   Choose the one with **best score** (e.g., lowest AIC). Suppose omitting X_4 is best. Now the model is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_5 X_5$$

3. Try removing one of the 4 remaining predictors:

   - $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3$
   - $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_5 X_5$
   - $Y = \beta_0 + \beta_1 X_1 + \beta_3 X_3 + \beta_5 X_5$
   - $Y = \beta_0 + \beta_2 X_2 + \beta_3 X_3 + \beta_5 X_5$

Repeat until stopping rule is triggered (e.g., no further improvement, or reaching a minimum number of variables).

⚠ **Assumptions and Limitations**

The Backward Stepwise Selection **cannot be used if $p > n$ (more variables than observations**), since the full model cannot be fit. Also, the algorithm is Greedy, so when **once a variable is remove, it's gone forever**, even if it would be useful with a different combination.

Linear regression solves for:

$$\hat{\boldsymbol{\beta}} = \left(X^T X\right)^{-1} X^T y$$

Where:

- X is the **design matrix**: it has n rows (observations) and p columns (predictors).

- $\hat{\beta}$ is the vector of estimated coefficients.

- $\left(X^T X\right)^{-1}$ is the **inverse** of the $p \times p$ matrix $X^\top X$

But this formula only works if $X^T X$ is invertible.

⚠ **The problem.** An invertible matrix must be **full-rank**. Also, a matrix $X \in \mathbb{R}^{n \times p}$ can have:

$$\text{at most rank} = \min\left(n, p\right)$$

Since the size of the inverse matrix is determined by that of the design matrix and its transpose $X^T \in \mathbb{R}^{p \times n}$, we must ensure that the rank of the design matrix is equal to or less than that of its inverse.

For example, let's say we have:

- $n = 2$ observations

- $p = 3$ predictors

So X might look like:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad \text{(2 rows, 3 columns)}$$

Then:

$$X^\top X = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 17 & 22 & 27 \\ 22 & 29 & 36 \\ 27 & 36 & 45 \end{bmatrix} \Rightarrow \text{a 3×3 matrix with rank} \leq 2$$

The rank is less than or equal to two because the rank of a matrix product cannot exceed that of its original matrices. More precisely:

$$\text{rank}\left(X^T X\right) \leq \text{rank}(X)$$

So if $p > n$, then the product $X^T X$ is **singular** and **not invertible** (because $\text{rank}(X) = \text{rank}\left(X^T X\right)$ and it is not full rank for $X^T X$), and the regression coefficients are **not computable**:

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T y$$

⚖️ Forward vs Backward

Feature	Forward Stepwise	Backward Stepwise
Starts with	No predictors	All predictors
Adds / Removes	Adds one at a time	Removes one at a time
Requires full model fit?	No	Yes (so needs $p \leq n$)
Greedy?	Yes	Yes
Typical usage	When p is large	When p is small ($p < n$)

5.4.6   Shrinkage Methods

In **Shrinkage Methods**, we **penalize the size** of the regression coefficients. Instead of minimizing only the **residual sum of squares (RSS)**, we minimize:

$$\text{RSS} + \lambda \cdot \text{Penalty}$$

- **RSS** ensures a good fit to the data.

- The **penalty** prevents overfitting by shrinking the coefficients **toward 0**.

- The **tuning parameter** $\lambda \geq 0$ controls the amount of shrinkage.

So:

- If $\lambda = 0$: no shrinkage, then same as ordinary least squares (OLS).

- If $\lambda \to \infty$: coefficients shrink toward 0.

🟢 Geometric View

In OLS, we find the point that minimizes RSS (ellipse contour). In shrinkage, we **constrain the solution** to be within a certain shape (circle, diamond, etc.). The Shrinkage is the **constrained optimization**.

❓ Why Shrink?

Benefit	How Shrinkage Helps
Reduce variance	Shrinking coefficients prevents wild oscillation.
Handle multicollinearity	Penalty makes solution stable when predictors are correlated.
Interpretability	Lasso even sets some coefficients exactly to 0.
Works when $p > n$	OLS fails, shrinkage still works.

The **General Optimization Form** for all shrinkage methods is:

$$\min_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \cdot \text{Penalty}(\beta) \right\} \tag{101}$$

- The first term is **RSS**.

- The second term is a **penalty** on the size of β.

- The **form of the penalty** determines the method.

5.4.6.1 Ridge Regression

Ridge Regression is a linear regression technique that adds an **L2 penalty** to the loss function (page 147) in order to **shrink** the regression coefficients and reduce model complexity. It is especially useful when the predictors are **highly correlated** or when there are **more predictors than observations** ($p > n$).

⚠ **What problems does ordinary linear regression have?**

In **linear regression**, we find the best coefficients β_j to predict Y from X:

$$\hat{\beta}^{\text{OLS}} = \arg\min_{\beta} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \arg\min_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

We're just minimizing the total squared error between what the model predicts and the actual data. But when predictors are **highly correlated**, or there are **many predictors**, or $p > n$ (more variables than observations), then OLS estimates become:

- ✖ **Unstable**: coefficients bounce around a lot.

- ✖ **Large**: some β_j become huge to compensate.

- ✖ **Overfit**: fits training data well, but fails on new data.

✅ **Ridge Regression: the fix.** The key idea is: let's stop the coefficients from growing too large. So we modify the loss function:

$$\text{Ridge Loss} = \text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

We still try to make predictions accurate (low RSS), **but now we also penalize large β_j** values.

Mathematically, Ridge Regression is a modified version of linear regression, it estimates the coefficients $\hat{\beta}$ by solving:

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{102}$$

- The first term:

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

  is the **residual sum of squares (RSS)**.

- The second term:

$$\lambda \sum_{j=1}^{p} \beta_j^2$$

  is the **ridge penalty**. It is the sum of squared coefficients.

- $\lambda \geq 0$ is the **tuning parameter** that controls the strength of shrinkage.

When $\lambda = 0$, Ridge Regression is equivalent to ordinary least squares (OLS). As λ increases, the coefficients are increasingly **shrunk toward zero**, but never exactly zero.

❓ What does L2 penalty mean?

In Ridge Regression, the **L2 penalty** refers to the **sum of the squares of the regression coefficients**:

$$\text{L2 penalty} = \sum_{j=1}^{p} \beta_j^2 = \|\boldsymbol{\beta}\|_2^2 \tag{103}$$

It's called **L2** because it's based on the **L2 norm** (also known as the Euclidean norm). The term $\|\boldsymbol{\beta}\|_2$ is the **distance** from the origin in coefficient space:

$$\|\boldsymbol{\beta}\|_2 = \sqrt{\beta_1^2 + \beta_2^2 + \cdots + \beta_p^2}$$

So the **L2 penalty** is just the **square** of that distance:

$$\|\boldsymbol{\beta}\|_2^2 = \beta_1^2 + \beta_2^2 + \cdots + \beta_p^2$$

❓ Why square the coefficients?

Because squaring, the penalty:

- **Grows fast** and it **heavily punishes large values** (for example, $10^2 = 100$).

- Is **smooth and differentiable**, making it easy to optimize.

- Makes the penalty **zero** when all $\beta_j = 0$.

So the model is **encouraged** to fit the data well and **keep coefficients small** (unless they're really helpful).

❓ What does λ do?

The λ term is called **weight** (or **tuning parameter**) and controls how much importance we give to the **penalty** compared to the **prediction error**. In other words, it is a balancing weight between fit (RSS) and simplicity (ridge penalty).

Value of λ	What happens
$\lambda = 0$	No penalty, same as OLS (possibly overfitting)
Small λ	Light shrinkage, keeps fit close to OLS
Medium λ	Shrinks coefficients significantly, improves generalization
Large λ	Strong penalty, forces all $\beta_j \to 0$, risk underfitting
$\lambda \to \infty$	Model ignores all predictors, then predicts the mean

❷ So, how do we choose the best λ?

We **don't guess it manually**. Instead, we use **cross-validation**:

1. Split our data into training and validation sets;

2. Try many values of λ;

3. For each, compute prediction error on validation data;

4. Choose the λ that gives the **lowest validation error**.

We use cross-validation because we want to find the value of λ that gives the best prediction performance on new, unseen data (and cross-validation simulates that).

❷ When should we use ridge regression, and when should we not?

1. We have **many predictors**, especially when $p \sim n$ or $p > n$. Because OLS becomes unstable and Ridge can controls the complexity.

2. Our predictors are **highly correlated (multicollinearity)**. Because OLS coefficients can become huge and erratic. Ridge **shrinks correlated coefficients smoothly**, improving stability.

3. We want **better prediction accuracy**, not interpretability. Ridge keeps **all variables**, it doesn't eliminate any. Great when our goal is **prediction**, not explanation.

4. We **don't need variable selection**. Ridge shrinks all coefficients, but **never sets them exactly to 0**. So it's not ideal when we want to identify the most important variables.

Don't use Ridge if:

✖ We want to **remove irrelevant features**, then use Lasso.

✖ We want a model that's **easy to interpret**, then Ridge keeps everything in.

5.4.6.2   Lasso

Lasso Regression (Least Absolute Shrinkage and Selection Operator)
is a linear regression method that adds an **L1 penalty** to the loss function. This
penalty encourages sparsity by shrinking some coefficients **exactly to zero**,
effectively performing **variable selection**.

Mathematically, Lasso solves:

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{104}$$

Where:

- The first term

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

  is the **Residual Sum of Squares (RSS)**.

- The second term

$$\lambda \sum_{j=1}^{p} |\beta_j|$$

  is the **L1 penalty**. It is the sum of the absolute values of the coefficients.

- $\lambda \geq 0$ controls the strength of the penalty.

❷ What is the L1 Penalty?

The **L1 penalty** is the **sum of the absolute values of the regression
coefficients**:

$$\text{L1 penalty} = \sum_{j=1}^{p} |\beta_j| = \|\boldsymbol{\beta}\|_1 \tag{105}$$

This is called the **L1 norm** of the coefficient vector. It's simply a measure of
how large the coefficients are, in terms of their **total absolute magnitude**.
The L1 penalty **set coefficients to exactly zero** because the **absolute value
function has a sharp corner at 0**, the optimizer finds ti "cheap" to **set
coefficients to zero** to reduce penalty. This is what allows **Lasso to do
variable selection**. In other words, it allows to **select important variables**
and discard the rest.

Penalty Type	Formula	Behavior
L2 (Ridge)	$\sum \beta_j^2$	Shrinks all coefficients smoothly
L1 (Lasso)	$\sum |\beta_j|$	Can shrink some to **exactly 0**

❷ Role of λ

Just like Ridge:

- $\lambda = 0$: Lasso is OLS.

- Larger λ: more shrinkage.

- **But with Lasso**, some coefficients become **exactly 0** when λ is big enough.

So Lasso not only regularizes but also **drops unimportant variables**. So during training, it **assigns a value of exactly zero to some coefficients**.

In other words, Lasso is like **automatic feature selection**. It keeps only the most important predictors, and discards the rest by setting their coefficients to zero. This is something Ridge Regression **cannot do**, because it **shrinks** coefficients, but **never eliminates them**.

Example 11: Lasso

Suppose we have 5 predictors:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \epsilon$$

After fitting Lasso, we might get:

$$\hat{\beta}_1 = 3.1, \quad \hat{\beta}_2 = 0, \quad \hat{\beta}_3 = 0, \quad \hat{\beta}_4 = 2.5, \quad \hat{\beta}_5 = 0$$

This means:

- X_1 and X_4 are **kept**.

- X_2, X_3, and X_5 are **removed** from the model.

---

❷ Why use Lasso?

- To reduce **overfitting** by controlling model complexity.

- To get a **simpler, more interpretable model**.

- To automatically **select important variables**.

- Works well when **some predictors are irrelevant**.
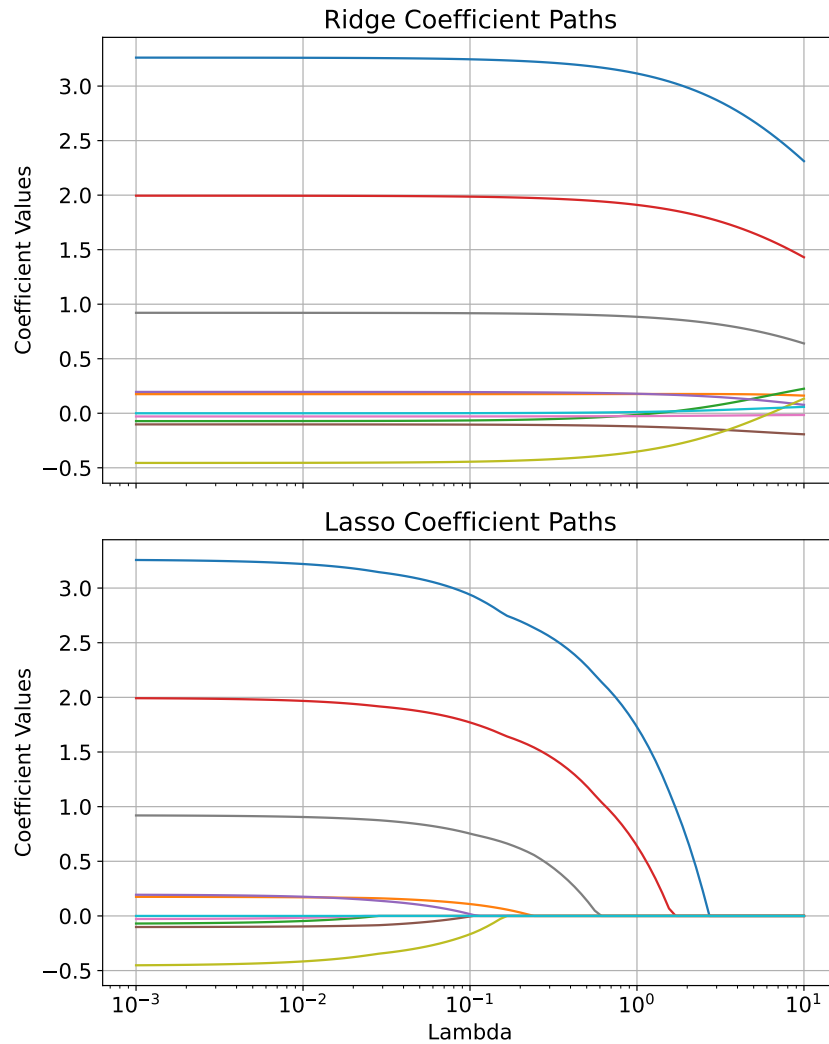
Figure 28: A visual comparison between **Ridge** and **Lasso** regression.

- **Ridge** (top): As λ increases, all coefficients **shrink smoothly toward zero**, but **none become exactly zero**.

- **Lasso** (bottom): As λ increases, some coefficients **shrink to zero** and stay there, the model **drops predictors** entirely.

This shows how Ridge keeps all variables but controls their influence, while Lasso both regularizes and performs **feature selection**.

5.4.6.3 Elastic Net

Elastic Net is a regression (hybrid) method that combines the strengths of **Ridge** and **Lasso** by using **both the L1 and L2 penalties**. It encourages **sparsity** like Lasso, and stabilizes the model like Ridge. The loss function is:

$$\hat{\beta}^{\text{EN}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \right\} \quad (106)$$

Where:

- The first term

$$\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j \right)^2$$

  is the **Residual Sum of Squares (RSS)**.

- The second term

$$\lambda \sum_{j=1}^{p} |\beta_j|$$

  is the **L1 penalty** (**Lasso**). It is the sum of the absolute values of the coefficients.

- The third term

$$\lambda \sum_{j=1}^{p} \beta_j^2$$

  is the **L2 penalty** (**Ridge penalty**). It is the sum of squared coefficients.

Sometimes it's written more compactly as:

$$\text{RSS} + \lambda \left(\alpha \sum |\beta_j| + (1 - \alpha) \sum \beta_j^2 \right) \quad (107)$$

Where:

- $\lambda \geq 0$: total regularization strength

- $\alpha \in [0, 1]$: balance between Lasso (L1) and Ridge (L2).

  - $\alpha = 0$: Elastic Net becomes the **Ridge** method.
  - $\alpha = 1$: Elastic Net becomes the **Lasso** method.
  - $0 < \alpha < 1$: Elastic Net is a mix of both.

❓ Why use Elastic Net?

- We have **many correlated predictors**.

- Lasso selects **only one** from a group of correlated variables and ignores the rest.

- We want **sparse output** (like Lasso) but also **grouped selection** and **stability** (like Ridge).

Feature	Lasso	Ridge	Elastic Net
Shrinks coefficients	✅	✅	✅
Sets coefficients to 0	✅	❌	✅
Handles correlated predictors	❌ (selects one)	✅ (spreads weight)	✅ (can select group)

❓ When to use which?

If we care about...	Use
Prediction accuracy & multicollinearity, keep all predictors	**Ridge**
Simplicity, variable selection, interpretability	**Lasso**
Both: want sparsity **and** handle correlation well	**Elastic Net**

6   GLMs and Logistic Regression

6.1   Introduction

A **Linear Model** (e.g., Linear Regression, page 101) is a very simple rule to *predict* a value Y (like someone's weight, income, house price, etc.). It says: take some **input variables** (like age, income, years of education), multiply each by a number (the **beta**), add them up, we get our **prediction**.

So for the i-th person:

$$Y_i = X_i^T \beta + \varepsilon_i$$

- X_i means all the **inputs** for person i (e.g., height, age, income), all together in a list (a vector).

- β is a set of **weights** telling us how important each input is.

- ε_i is the **random error**, the difference between what we predict and what we really see.

When we learned Ordinary Least Squares (OLS, page 102) regression, we saw:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \cdots + \varepsilon_i$$

That is a linear model. It predicts a continuous outcome (like price, weight, score). It fits the coefficients β by minimizing the squared differences between the predicted and actual values.

❷ Okay, but what's the point?

In other words, the model says *on average*, the value we expect for Y_i is:

$$E[Y_i] = X_i^T \beta$$

So, for a normal linear regression, we believe the average outcome **IS** just a line (or plane) in our input space.

⚠ So what is the problem?

The problem is simple. Imagine we use this same idea to predict whether someone buys a product: yes (1) or no (0). If we use a line, it can predict values like 1.2 or -0.3, **nonsense**, because we can't have a probability bigger than 1 or smaller than 0.

So, a standard linear model is **good for numbers that can be any real number** (like income), but bad for things like:

- Probabilities, must be in $[0, 1]$

- Counts, must be ≥ 0

And **that's where Generalized Linear Models (GLMs) comes in**. They generalize Linear Regression to handle: Binary data (Logistic Regression), Count data (Poisson Regression) and other special cases.

6.2   Generalized Linear Model (GLM)

A **Generalized Linear Model (GLM)** is a flexible extension of the standard linear regression that models the **mean** of a response variable Y through a **link function**. A GLM has three key components:

- **Random component**. The response variable Y follows a distribution from the **exponential family** (e.g., normal, Bernoulli, Poisson).

- **Systematic component**. Predictors X_i are combined linearly:
$$\eta_i = X_i^T \beta$$

- **Link function**. A function $g(\cdot)$ relates the expected value of Y to the linear predictor:
$$g(E[Y_i]) = \eta_i = X_i^T \beta$$

In short, a GLM is any model where:

- The **mean** is not modeled directly, but *through a link*.

- The data can follow various distributions (not just normal).

- Fitting is done by Maximum Likelihood, not just OLS.

❓ Why do we need Generalized Linear Models (GLMs)?

As we saw in the previous section, the key problem with linear regression is that we model:
$$E[Y_i] = X_i^T \beta$$

But **some types of data don't fit this**.

❓ **So what's the big idea of GLMs?** **Don't model** $E[Y_i]$ **directly**. Instead, model a **transformation** of it, so it stays inside valid limits. This is called the **link function**, $g(\cdot)$:
$$g(E[Y_i]) = X_i^T \beta$$

> **Example 1: Binary Data**
>
> Suppose:
> $$Y_i \in \{0, 1\}$$
>
> Maybe it's: *did the custom buy?* yes (1) or no (0). The thing we want to predict is a **probability**, which must be between 0 and 1. If we use a simple line:
> $$P(Y_i = 1) = X_i^T \beta$$
>
> This can easily give us values like 1.2 or -0.1, **impossible** for a probability.

We use the logistic link:

$$g(p) = \log\left(\frac{p}{1-p}\right)$$

So instead of saying:

$$P(Y_i = 1) = X_i^T \beta$$

We say:

$$\log\left(\frac{P(Y_i = 1)}{1 - P(Y_i = 1)}\right) = X_i^T \beta$$

This forces the probability to stay between 0 and 1. This is **Logistic Regression**.

6.2.1   Link Functions

A **Link Function** connects the mean of our response variable Y to the linear predictor (the same linear piece as in ordinary linear regression).

A **link function** $g(\cdot)$ is a mathematical function such that:

$$g(E[Y_i]) = X_i^T \beta \tag{108}$$

So instead of predicting the mean **directly**, we predict a **transformation** of it.

❷ Why is it essential?

Because for some data types (like binary or counts), the mean must lie in a specific range. The link function **ensures** that the model output stays inside valid bounds.

☰ Types of link functions

- **Identity link** (Linear Regression). It is used in standard linear regression and is considered naïve because it **does not make any transformations**:

$$g(\mu) = \mu \tag{109}$$

- **Logit Link** (Logistic Regression). It is used for **binary outcomes**, because we need the mean (the probability) to stay in $(0, 1)$. So we model the **log-odds**:

$$g(\mu) = \log\left(\frac{\mu}{1-\mu}\right) \tag{110}$$

This stretches the probability μ from $(0, 1)$ to the whole real line $(-\infty, \infty)$. So any value of $X^T \beta$ makes sense.

❷ **Why take the log?** Because the **odds** are always positive (0 to infinity), and the **log-odds** stretch this to the whole real line $(-\infty, +\infty)$.

  - If odds $= 1 \rightarrow$ log-odds $= 0$.
  - If odds $< 1 \rightarrow$ log-odds negative.
  - If odds $> 1 \rightarrow$ log-odds positive.

So **log-odds** can be any real number and it is perfect for linear modeling!

Remark: *What are odds and log-odds?*

Suppose:

$$p = P(Y = 1) = 0.8$$

So we have an 80% chance of success. The **odds** of success are:

$$\text{odds} = \frac{p}{1-p} = \frac{0.8}{0.2} = 4$$

So we are 4 times more likely to succeed than to fail.

In other words, the **odds are the ratio of success to failure**.

The **log-odds** is just:

$$\text{log-odds} = \log(\text{odds}) = \log\left(\frac{p}{1-p}\right) \tag{111}$$

- **Log Link** (Poisson Regression). It is used for counts, where the mean must be ≥ 0:

$$g(\mu) = \log(\mu) \tag{112}$$

So:

$$\log(\lambda) = X^T\beta \quad \implies \quad \lambda = e^{X^T\beta} > 0 \tag{113}$$

❓ What do we really model in a GLM?

In a **GLM**, we do **not** directly model the mean $E[Y]$. Instead, we model the **result of applying the link function** to the mean. Formally:

$$g(E[Y]) = X^T\beta$$

So **the actual linear model** is for:

$$\underbrace{g(E[Y])}_{\text{link of the mean}}$$

In other words, we choose a link function $g(\cdot)$ to *transform* the mean of Y so it can be modeled linearly. Then we fit the linear predictor to that **transformed mean**. The output of the link is **not** the raw mean, but a transformed version of it.

In a GLM, we always model the link of the mean, not the mean itself.

6.2.2   Inverse Link Function

In a **GLM**, we do not model the mean $E[Y]$ directly, we model **the link of the mean**:

$$g(E[Y]) = X^T \beta$$

To get the actual mean back on its **natural scale**, we apply the **Inverse Link Function**:

$$E[Y] = g^{-1}(X^T \beta) \tag{114}$$

❓ Why it matters

- The **link function** transforms the mean to make a linear model possible.

- The **inverse link** undoes this transformation so we get predictions in the real-world units we care about.

✊ We are still not convinced. Why do we need the original scale? It's simply because we **can't interpret log-odds intuitively**. If we tell a client "the log-odds is 1.4", no one knows what that means. But if we tell them "the probability of success is 80%", that's meaningful. So the workflow is:

1. Model the **link of the mean** linearly (e.g., $\text{logit}(p) = X^T \beta$).

2. Estimate β by fitting this transformed model.

3. Use the **inverse link** (e.g., sigmoid) to transform the result back. It gives valid probability.

4. Make decisions with the actual probability!

In other words, the link is for the math, and the inverse link is for meaningful output. If we never invert the link we only have the transformed mean which by itself is useless for practical interpretation.

☰ Types of inverse link functions

- **Identity Link** (Linear Regression). Its link is:

$$g(\mu) = \mu$$

And the inverse link function is:

$$g^{-1}(z) = z \tag{115}$$

Nothing changes! The result is: $E[Y] = X^T \beta$.

- **Logit Link** (Logistic Regression). Its link is:

$$g(p) = \log\left(\frac{p}{1-p}\right)$$

And the inverse link function is:

$$g^{-1}(z) = \frac{1}{1 + e^{-z}} \quad (\textbf{Sigmoid}) \tag{116}$$

The result is:

$$P(Y = 1|X) = \text{sigmoid}(X^T \beta)$$
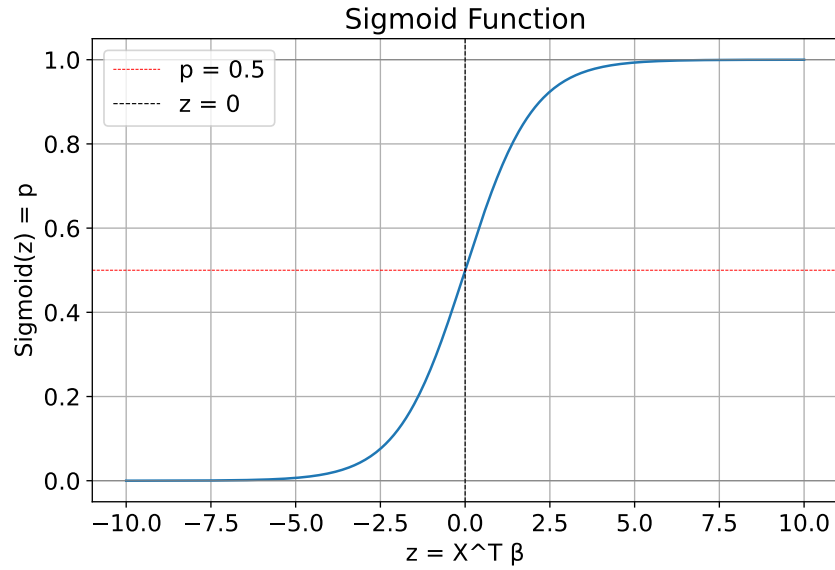
Figure 29: Example of **sigmoid curve**. We can see its classic S-shape.

- For large negative z, the output p stays close to 0.
- For large positive z, the output p stays close to 1.
- At $z = 0$, it crosses $p = 0.5$.

So no matter how large or small our linear predictor $z = X^T\beta$ is, the sigmoid safely squashes it into a valid probability.

- **Log Link** (Poisson Regression). Its link is:

$$g(\lambda) = \log(\lambda)$$

  And the inverse link function is:

$$g^{-1}(z) = e^z \quad \text{(exponential)} \tag{117}$$

  The result is:

$$E[Y] = \lambda = \exp(X^T\beta)$$

So **every link function has an inverse**. The link is for safe modeling, and the inverse gives us the actual prediction in meaningful units.

6.2.3   GLMs for Count Data - The Poisson Model

Suppose we want to model **counts** (e.g., number of emails per day, number of accidents per month, number of clicks per ad). The response variable Y_i is must be a **non-negative integer**: 0, 1, 2, etc.

The standard model for counts is the **Poisson Distribution**:

$$Y_i \sim \text{Poisson}(\lambda_i) \tag{118}$$

The **mean** of Y_i is:

$$E[Y_i] = \lambda_i \quad (\lambda_i > 0) \tag{119}$$

✅ Link function and invert link function

We need to force $\lambda_i > 0$. So we use the **log link**:

$$g(\lambda_i) = \log(\lambda_i) = X_i^T \beta \tag{120}$$

To get the mean back, invert the link:

$$\lambda_i = e^{X_i^T \beta} \tag{121}$$

So the GLM for counts is:

- **Random part**:

$$Y_i \sim \text{Poisson}(\lambda_i)$$

- **Systematic part**:

$$\eta_i = X_i^T \beta$$

- **Link function**:

$$\log(\lambda_i) = \eta_i \quad \Longrightarrow \quad \lambda_i = \exp(\eta_i)$$

❓ What does this do?

Ensures our predicted mean is always positive. Allows the **multiplicative effect**: a one-unit change in X_j multiplies λ by e^{β_j}. For example, if $\beta_j = 0.1$, then a one-unit increase in X_j **multiplies the expected count** by $e^{0.1} \approx 1.105$. So we get about a 10.5% **increase** in the expected count.

6.2.4   Fitting GLMs

In **Linear Regression**, we find β by minimizing the **Sum of Squared Errors**, that's the famous **Ordinary Least Squares (OLS)**. But for **GLMs**, OLS **does not work** because:

- The error distribution is not normal anymore.

- The mean is transformed by a link function.

- We need to respect the correct likelihood for the chosen distribution.

So **GLMs are fit by Maximum Likelihood Estimation (MLE)**.

❷ **How does Maximum Likelihood work?**

Imagine we have data, like:
$$(Y_1, Y_2, \ldots, Y_n)$$

We assume:

- The data points are **random**, but they follow some probability model that depends on parameters β.

- Our goal is to find the **best** β.

But how do we decide what "best" means? The "best" parameters are the ones that make the data we actually observed as **likely** as possible. If our model is good, the **probability of our actual data should be high**. So we pick β that **maximizes** that probability.

1. **Write the probability of one point**. For example, for Logistic Regression:
$$Y_i \sim \text{Bernoulli}(p_i), \quad p_i = \text{sigmoid}(X_i^T \beta)$$

   So:
$$P(Y_i \mid X_i, \beta) = p_i^{Y_i}(1 - p_i)^{1 - Y_i}$$

2. **Multiply them all together**. For the whole dataset:
$$L(\beta) = \prod_{i=1}^{n} P(Y_i \mid X_i, \beta)$$

   This is the **likelihood function**.

3. **Take the log**. Multiplying many tiny probabilities is numerically annoying. So we take the **log-likelihood**:
$$\ell(\beta) = \sum_{i=1}^{n} \log P(Y_i \mid X_i, \beta)$$

   The maximum is in the same place because the log is monotonic.

4. **Maximize the log-likelihood**. Find the β that gives the highest $\ell(\beta)$. In other words, find the β that makes the data we observed **most probable** under our model.

❷ **How do we do the maximization?** We take the **derivate** of the log-likelihood (*score function*):

$$U(\beta) = \frac{\partial \ell(\beta)}{\partial \beta} \tag{122}$$

We solve $U(\beta) = 0$. In practice, we can't solve this exactly, we do it **numerically**:

- Start with a guess for β
- Compute the gradient (the slope)
- Step in the direction that increases the likelihood
- Repeat until the slope is flat (maximum found)

This is like climbing a hill until we reach the top. It works by iteratively solving the score equations until the likelihood is maximized.

⚖️ How is this different from OLS?

In OLS, we don't think in terms of probabilities. We just minimize the sum of squared differences:

$$\text{SSE} = \sum_{i=1}^{n}(Y_i - X_i^T\beta)^2$$

This *is* MLE **when** the errors are normal. If we assume Y is Normal, the MLE for the mean is exactly the same as minimizing SSE. But for **binary data**, **counts**, the Normal is wrong, so SSE doesn't make sense. MLE naturally picks the correct probability model.

In conclusion, OLS is a special case of MLE when the errors are Normal. When the errors follow Bernoulli, Poisson, etc., we use the **correct likelihood**, not squared errors. MLE finds the parameter values that make our data "least surprising" under our chosen probability model.

> **Key Takeaways: Fitting GLMs**
>
> - For **linear regression**: find β that **minimize sum of squared errors**.
>
> - For **GLMs**: find β that **maximize the likelihood**, because we now have a proper probability model for our special data.

6.2.5   Exponential Family

The **Exponential Family** is a big **family of probability distributions that share a convenient mathematical form**.

❷ **Why do we care?** Because GLMs are **built** for any distribution in this family, the math works cleanly and link functions are easy to define.

🗒 Canonical form

A distribution belongs to the **exponential family** if its probability density (or mass) function can be written as:

$$f(y; \theta, \phi) = \exp\left(\frac{y\theta - b(\theta)}{a(\phi)} + c(y, \phi)\right) \tag{123}$$

Where:

- θ: the **canonical (natural) parameter**.
- $b(\theta)$: controls the mean.
- $a(\phi)$: the **dispersion parameter** (like variance).
- $c(y, \phi)$: normalizes it.

Canonical form is a standardized, structured way to write the probability distribution from the exponential family. It is like a "template" that all members of the exponential family can fit into.

❷ Why we use this form

This form has standard formulas for the mean and variance:

- The **mean**: $E[Y] = b'(\theta)$
- The **variance**: $\text{Var}(Y) = b''(\theta)\, a(\phi)$

So they're easy to work with! Also, it is useful for GLMs because it assumes our data comes from an exponential family distribution. This **guarantees there's a link function that connects the mean to the linear predictor**.

This is the theoretical backbone of GLMs: every GLM assumes the response comes from the exponential family.

Distribution	Random variable	Canonical parameter θ	Common link
Normal	$Y \in \mathbb{R}$	μ	Identity
Bernoulli	$Y \in \{0, 1\}$	$\log\frac{p}{1-p}$	Logit
Poisson	$Y \in \{0, 1, 2, ...\}$	$\log\lambda$	Log

Table 17: Examples of exponential family members.

6.3 Logistic Regression

6.3.1 What is Logistic Regression?

Logistic Regression is a **Generalized Linear Model (GLM)** used to model a **binary outcome** (i.e., when the response Y can only be 0 or 1). It models the **probability** that $Y = 1$ as a **sigmoid function** of a linear combination of predictors. Formally:

$$Y_i \sim \text{Bernoulli}\,(p_i) \quad \text{where} \quad p_i = P\,(Y_i = 1|X_i) \tag{124}$$

So we have an observation i with input data X_i. The outcome Y_i can be 0 or 1. The **probability** that $Y_i = 1$ is p_i. This means that the random variable Y_i follows a Bernoulli distribution:

- $Y_i = 1$ with probability p_i.

- $Y_i = 0$ with probability $1 - p_i$.

So we say: "The *distribution* of Y_i is Bernoulli, with success probability p_i".

❓ How is p_i determined?

We don't just guess p_i! We **model it** using our predictors X_i and some coefficients β. But we can't model p_i directly with a linear predictor because probabilities must stay in $[0, 1]$.

So we **transform** p_i using the **logit link** (page 180):

$$\log\left(\frac{p_i}{1 - p_i}\right) = X_i^T \beta \quad (\textbf{Logit Link})$$

Where:

- The expression inside the log is called the **odds**:

$$\text{odds} = \frac{p}{1 - p}$$

- The **log of the odds** (log-odds) can be any real number, so it is perfect for linear modeling.

- So we model **log-odds** as a linear function of our predictors:

$$\text{log-odds of success} = X_i^T \beta$$

This means that if $X_i^T \beta$ goes up, the log-odds go up. And bigger log-odds, higher probability of success.

↻ **Why is the *sigmoid* version necessary?**

We've already seen why Inverse Link Functions exist (see page 182). However, the *logit* equation shows how we model the probability p_i. But to get back to the **actual probability**, we invert the *logit* using the **Sigmoid function**:

$$p_i = \frac{1}{1 + e^{-X_i^T \beta}} \quad \textbf{(Sigmoid)}$$

In simple terms, take our linear combination $X_i^T \beta$, plug it into the **sigmoid function**, and we get p_i, which is guaranteed to be between 0 and 1. So the sigmoid "squashes" the real line to the unit interval.

> **Key Takeaways**
>
> So the big picture is:
>
> - The **logit form** is how we **build** and **fit** the model.
>
> - The **sigmoid form** is how we **interpret** it and **make probability predictions**.
>
> They are just two sides of the same coin.

6.3.2 How does Logistic Regression fit?

As we saw with the Generalized Linear Models (GLMs, page 185), Logistic Regression **cannot** be fitted by ordinary least squares (OLS) because:

- The error distribution is not normal anymore.

- The mean is transformed by a link function.

- We need to respect the correct likelihood for the chosen distribution.

Instead, we use **Maximum Likelihood Estimation (MLE)** (see page 185 for an explanation of how MLE works).

❷ Goal: How do we find the best line?

In **Linear Regression**, we find the line that **minimizes squared errors (OLS)**. That works because the data are continuous and the errors are Normal. But in Logistic Regression our outcome is 0 or 1, so the normal error idea makes no sense. The predictions must be probabilities between 0 and 1, so we can't just draw any line.

So instead, we use **Maximum Likelihood Estimation (MLE)**, that find the line (the coefficients β) that make the observed 0s and 1s **most probable** under the model. Where "most probable" means that the model's predicted probabilities match what we actually observed **as closely as possible**, in terms of probability.

Each observation is formally defined as follows:

$$Y_i \sim \text{Bernoulli}(p_i), \quad p_i = \text{sigmoid}(X_i^T \beta)$$

For each data point i the outcome Y_i is either 0 or 1, and the probability that $Y_i = 1$ is p_i. So Y_i follows a Bernoulli distribution with probability p_i. In other words, each data point is assumed to be a Bernoulli trial, with probability given by the sigmoid of the linear prediction.

So the probability of observing Y_i (**probability of observing one data point**) is:

$$P(Y_i | X_i) = p_i^{Y_i} (1 - p_i)^{1 - Y_i}$$

That's a smart way to write:

$$\begin{cases} P(Y_i = 1 | X_i) = p_i & \text{if } Y_i = 1 \\ P(Y_i = 0 | X_i) = 1 - p_i & \text{if } Y_i = 0 \end{cases}$$

The **likelihood** for all data is the product:

$$L(\beta) = \prod_{i=1}^{n} p_i^{Y_i} (1 - p_i)^{1 - Y_i} \tag{125}$$

Where the data points are **independent**. So the probability of seeing *all* our data is the **product** of the single-point probabilities. It shows how likely our data is given the β of our model.

Finally, take the log to simplify (**log-likelihood**):

$$\ell(\beta) = \sum_{i=1}^{n} \left[Y_i \log\left(p_i\right) + (1 - Y_i) \log\left(1 - p_i\right) \right] \tag{126}$$

We take the log for simplification purposes:

- Products are annoying to handle.

- Logs turn products into sums.

- Sums are easier to differentiate.

So the log-likelihood is just the log of the likelihood, it **still measures the same fit BUT easier to work with**. Therefore, the <mark>higher the number, the better our model's β matches our observed 0s and 1s</mark>.

❓ **The trick is to find the best β. But how do we identify the *best* β?**

We find the β that makes $\ell(\beta)$ **as big as possible**. This is the same as finding the **maximum** of a curved hill. This is the **core idea** of MLE:

- Not "minimize distance" like OLS;

- But "maximize the chance we'd see this data if our model is true".

However, there's **no formula** like OLS. So (see page 185):

1. We start with a guess for β.

2. We check how well that guess "fits" (the likelihood).

3. We adjust β a little, so the overall match gets better.

4. We repeat, until the adjustments stop improving the fit.

This is done by **Gradient ascent** (move in the direction that increases fit) or **Newton-Raphson** or **Iteratively Reweighted Least Squares (IRLS)** (more efficient ways to jump toward the best spot).

After we find the best β (called $\hat{\beta}$), our fitted model is:

$$\log\left(\frac{\hat{p}}{1 - \hat{p}}\right) = X^T \hat{\beta} \quad \implies \quad \hat{p} = \frac{1}{1 + e^{-X^T \hat{\beta}}} \tag{127}$$

6.3.3   Practical Aspects

When we build a Logistic Regression, we must decide: ***which predictors (variables) should stay in our model?*** For example, maybe we start with 10 possible predictors, but, do they *all* matter? Or are some noise? Adding more predictors **always** increases our model's flexibility. But more predictors:

- Make our model more **complex**;

- Risk **overfitting**: we match our training data perfectly, but fail on new data.

We want a model that fits well without being overly complex. We balance this trade-off with AIC, BIC and Deviance.

❓ Why not just look at accuracy?

We could say: "*we'll just choose the model that predicts our 0s and 1s perfectly*". But if we keep adding variables, our model can memorize the training data. That **looks good on training data**, but it won't generalize. So accuracy alone can trick us!

📋 Variable Selection: AIC, BIC

In Logistic Regression, we often have **many predictors**, but not all of them may be useful. We want a model that is good at predicting and not unnecessarily complicated. Then **AIC (Akaike Information Criterion)** and **BIC (Bayesian Information Criterion)** help us choose (see page 156). Both balance:

- **Goodness of fit** (higher log-likelihood is better)

- **Penalty for model complexity** (more predictors = higher penalty)

$$
\begin{aligned}
\text{AIC} &= -2\ell\left(\hat{\beta}\right) + 2k \\
\text{BIC} &= -2\ell\left(\hat{\beta}\right) + k\log(n)
\end{aligned}
$$

Where:

- $\ell(\hat{\beta})$: maximized log-likelihood.

- k: number of estimated parameters.

- n: number of data points.

AIC and BIC help us decide **which predictors to keep** in our Logistic Regression model. They balance how well our model *fits* the data (log-likelihood) and how *complex* our model is (number of predictors).

❓ **AIC & BIC: How we use them.** Try different models (add or remove variables), calculate AIC and BIC for each, and pick the model with the **lowest AIC or BIC**. These are for **model selection**, not for predicting new labels directly.

📗 Regularization: Lasso, Ridge

When we have **many predictors**, we risk:

- **Overfitting**: model fits noise, not the signal.

- **Instability**: coefficients blow up, especially if predictors are correlated.

The solution is **add a penalty to shrink or select coefficients**.

- **Lasso (L1 penalty)** (see page 172). Adds a penalty for the absolute value of coefficients. Encourages some coefficients to shrink exactly to zero, it does **variable selection**.

$$\text{Penalized Objective:} \quad \ell(\beta) - \lambda \sum_{j=1}^{p} |\beta_j|$$

- **Ridge (L2 penalty)** (see page 169). Adds a penalty for the squared value of coefficients. Shrinks coefficients but does **not** set them to zero, then no variable selection, but **reduces variance**.

$$\text{Penalized Objective:} \quad \ell(\beta) - \lambda \sum_{j=1}^{p} \beta_j^2$$

λ is chosen by cross-validation. Both control model complexity, then we have a better generalization.

❌ No natural R^2

In Linear Regression, R^2 is the fraction of variance explained (see page 110):

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

But for Logistic Regression:

- ✘ There's no residual sum of squares because the **output is probability**.

- ✘ There's no direct "variance explained", because Logistic Regression does not model variance in a continuous sense.

Instead, we use:

- **Pseudo-R^2** (or **Pseudo-R-Squared**). It mimics the idea of R^2 but **uses log-likelihood** metrics instead. There are different versions of the pseudo R-squared. One of the most famous is the McFadden version:

$$R^2_{\text{McFadden}} = 1 - \frac{\ell(\text{model})}{\ell(\text{null})} \tag{128}$$

This tells us *roughly* **how much better our model is compared to the simplest model** (with no predictors). Pseudo-R^2 is **not** the same as OLS R^2. It's just a rough guide for fit.

Pseudo-R^2 is a **quick way** to gauge if our model has meaningful explanatory power at all.

- **Deviance** is like the sum of squared residuals in Linear Regression, but for GLMs. It's derived from the log-likelihood: **lower deviance means better fit**. It shows how well our model fits compared to a "null" model (with no predictors at all). A big drop in deviance means our predictors are doing real work. It's another tool to see if adding a variable actually improves fit enough to justify its cost.

  ❓ **How we use it:** Compare deviance of our model to a **null model** (which has no predictors). Big reduction in deviance means our **predictors help a lot**.

❓ **Why do we use them for Logistic Regression?**

- **AIC & BIC**: help pick the **best subset of variables**, balancing fit vs. complexity.

- **Deviance**: tells us if our predictors actually improve the model.

- **Pseudo-R^2**: gives us a rough single number for "model strength".

These help us **build and trust the model**, _before_ we check prediction performance with classification metrics. These tools are used to build and choose the best Logistic model. However, we need classification metrics to test how good our final model is at making decisions.

6.3.4 Interpreting Logistic Regression Coefficients

In Logistic Regression, each β_j **does not** directly tell us the change in probability like a slope in Linear Regression. Instead, **each β_j tells us how the log-odds change when that predictor increases by 1 unit**.

Formally, **Odds** is:

$$\text{odds} = \frac{p}{1-p} \quad \text{(ratio of success to failure)}$$

And **Log-Odds** (logit) is:

$$\text{log-odds} = \log\left(\frac{p}{1-p}\right)$$

The logistic regression model is naturally written in **log-odds**:

$$\text{log-odds} = X^T \beta$$

So β_j shows how much the **log-odds** change per unit increase in X_j. However, log-odds is not very intuitive for most people. For example, if $\beta_j = 0.5$, the log-odds goes up by 0.5, but what does that mean in real life? So we **exponentiate** the log-odds change, and we get the **odds ratio**. This turns an *additive* change on the log scale into a *multiplicative* effect on the odds.

If we exponentiate β_j, we get **Odds Ratio**:

$$\text{OR}_j = e^{\beta_j} \tag{129}$$

For each 1-unit increase in X_j, the **odds of success** multiply by OR_j. This is much easier to explain:

- "The odds double" (if OR = 2).

- "The odds increase by 50%" (if OR = 1.5).

- "The odds decrease by half" (if OR = 0.5).

References

[1] Herman Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American statistical Association*, 68(342):361–368, 1973.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer Series in Statistics. Springer New York, 2009.

[3] IBM Corporation. What is exploratory data analysis (eda)?, n.d. Accessed: 2025-06-26.

[4] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: with Applications in Python.* Springer Texts in Statistics. Springer New York, 2013.

[5] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis.* Applied Multivariate Statistical Analysis. Pearson Prentice Hall, 2007.

[6] Beraha Mario. Applied statistics. Slides from the HPC-E master's degree course on Politecnico di Milano, 2024-2025.

[7] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

Index

E

F

G

H

I

K

L

W