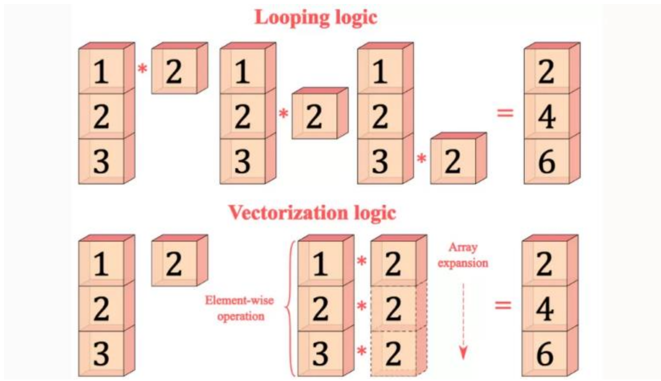
Why NumPy is Faster than Pure Python Loops?

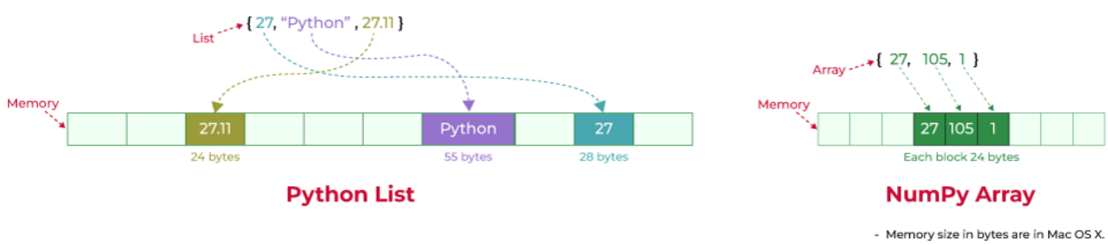
Using NumPy for large datasets (millions of samples) is significantly more efficient than pure Python loops for four main reasons:

1. **Vectorization:** NumPy performs operations on entire arrays at once instead of processing elements one by one. This eliminates the overhead of Python's for loops.

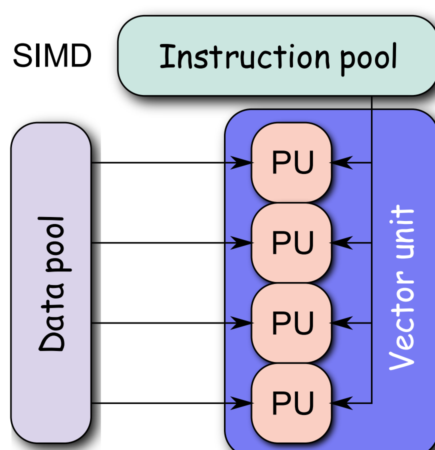


2. **Pre-compiled C Code:** Most NumPy operations are written in **C**, which is a low-level language. This allows for much faster execution near the computer's hardware limit, unlike Python, which is an interpreted language.

3. **Contiguous Memory:** NumPy stores data in a continuous block in memory. This allows the CPU to access data much faster (cache-friendly) compared to Python lists, which store data in scattered locations.

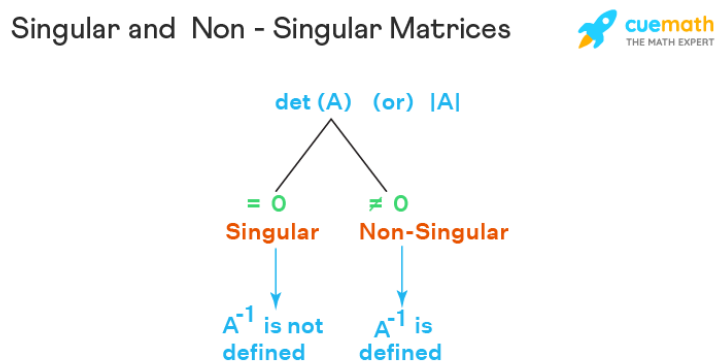


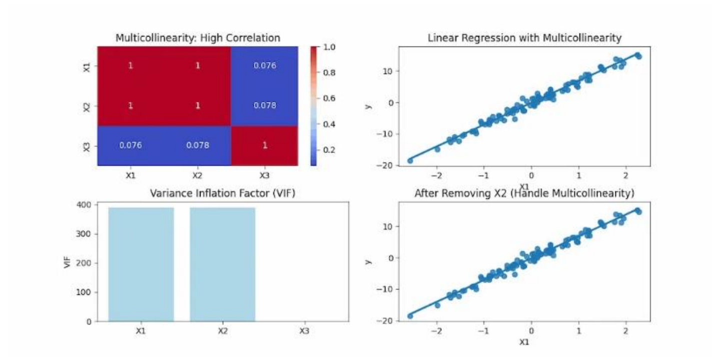
4. **Parallelism (SIMD):** Modern CPUs use **SIMD** (Single Instruction, Multiple Data) to perform the same calculation on multiple data points simultaneously. NumPy is designed to leverage this hardware feature, while standard Python loops are not.



1. The Mathematical Problem:

When features are highly correlated or duplicated, the matrix $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})$ becomes **Singular** (also known as **Non-Invertible**). This means its **Determinant is zero**, and mathematically, you cannot calculate its inverse $(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}$ Since the Normal Equation depends on this inverse, the formula fails to produce a result.





2. Why does this happen?

This happens due to **Linear Dependency**. If one feature is a multiple of another (e.g., house size in sq. ft. vs. sq. meters), they don't provide "new" information. In matrix algebra, this means the columns of your matrix X are not independent, causing the matrix to lose "Full Rank" and making the inverse impossible to compute.

3. The Solution (Without losing information):

The most effective solution is **Regularization**, specifically **Ridge Regression (L2 Regularization)**. Instead of removing features, we add a small "penalty" term $\boldsymbol{\lambda}$ to the diagonal of the matrix:

$$\boldsymbol{\theta} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \boldsymbol{\lambda}\boldsymbol{I})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

$$\frac{\partial J_{ridge}}{\partial \beta} = -\frac{1}{n}\mathbf{X}^{\top}(\mathbf{y} - \mathbf{X}\beta) + 2\lambda\beta = 0$$

Solving for β:

$$\mathbf{X}^{\top}\mathbf{X}\beta + n \cdot 2\lambda\beta = \mathbf{X}^{\top}\mathbf{y}$$

$$\beta = (\mathbf{X}^{\top}\mathbf{X} + 2n\lambda\mathbf{I})^{-1}\mathbf{X}^{\top}\mathbf{y}$$

- **How it works:** Adding this small value $\boldsymbol{\lambda}$ ensures that the matrix is no longer singular and will always have a valid inverse.
- **The Benefit:** It stabilizes the calculation and allows the model to handle correlated features without needing to delete them.