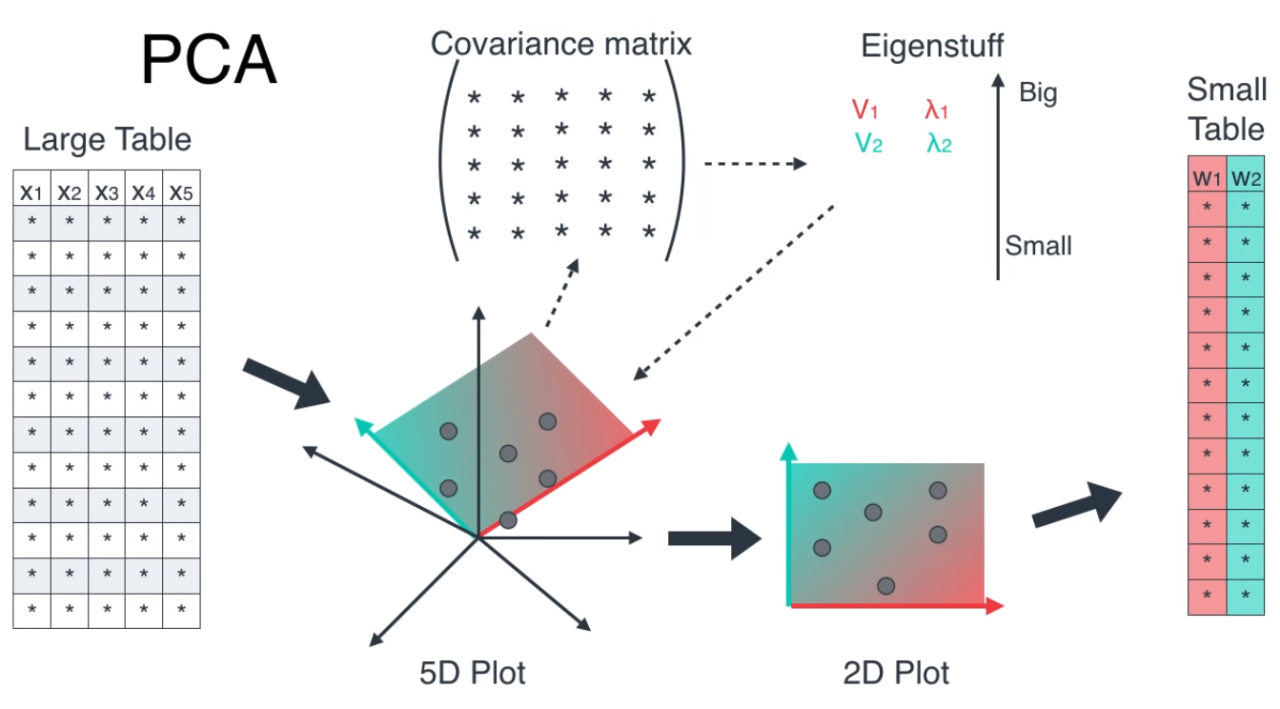
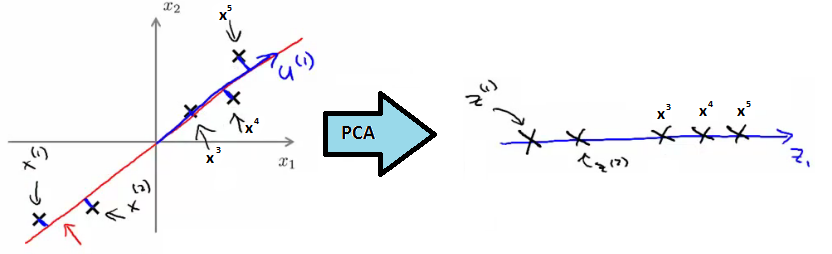
## **Module 8: Dimensionality Reduction – PCA**





<https://www.linkedin.com/pulse/principal-component-analysis-dimension-reduction-1-mukesh-manral>

<https://www.geeksforgeeks.org/machine-learning/mathematical-approach-to-pca/>

<https://www.geeksforgeeks.org/engineering-mathematics/eigen-values/>

### **Lesson Introduction**

When working with datasets having dozens or even hundreds of features, models can become slow, complex, and prone to overfitting. In such cases, **Dimensionality Reduction** techniques help us simplify the data without losing much important information.

### **Real-world Hook**

Think about a high-resolution image (e.g., 4000×4000 pixels). That's **16 million features**! If you could reduce it to just a few hundred important features without losing much visual quality, you could:

* Speed up processing
* Reduce storage
* Make algorithms faster and more accurate

This is exactly what **PCA** does.

### **Introduce Topic as a Solution**

**Dimensionality Reduction** refers to techniques that transform high-dimensional data into a lower-dimensional form, keeping the **most important information**.

**PCA (Principal Component Analysis)** is the most popular method used for:

* Visualization
* Noise reduction
* Speed-up of ML algorithms

## **Theory**

### **1. What is Dimensionality Reduction?**

Reducing the number of features while retaining the essential patterns and structures in the dataset.

**Why?**

* Avoid overfitting
* Reduce computational cost
* Improve visualization
* Remove multicollinearity

Dimensionality reduction is **a machine learning process that simplifies high-dimensional data by reducing the number of features while preserving essential information**, which combats issues like overfitting, computational complexity, and poor model performance. Techniques, such as Principal Component Analysis (PCA), transform data into a lower-dimensional space, making models easier to train, interpret, and visualize.

**Why it's needed:**

* **The Curse of Dimensionality**: High-dimensional data (with many features) is difficult for models to handle, leading to increased computational cost, more complex models, and a higher risk of overfitting.
* **Computational Efficiency**: Reducing the number of features significantly speeds up the model training process and reduces the storage requirements for large datasets. [
* **Improved Model Performance**: By eliminating irrelevant or redundant features, dimensionality reduction helps models generalize better to new data and avoid being misled by noise.
* **Data Visualization**: It's difficult to visualize data with many dimensions; reduction to 2 or 3 dimensions makes complex relationships in the data easier to understand.

**Common Techniques:**

* **Principal Component Analysis (PCA)**: A popular method that transforms the original features into a new set of uncorrelated components (principal components), ordered by the amount of variance they capture, allowing you to select the most important ones.
* **Feature Selection**: Methods that identify and keep only the most relevant features, effectively removing less important ones without transforming them.
* **Autoencoders**: A type of neural network used for learning a compressed, low-dimensional representation of data.
* **Independent Component Analysis (ICA)**: A linear technique that finds statistically independent components within data, rather than just uncorrelated ones.

**Challenges:**

* **Information Loss**: A trade-off exists between reducing dimensions and losing some critical information, which can impact model accuracy.
* **Interpretability**: Reducing features can sometimes make it harder to understand what the original variables represented.
* **Choosing the Right Technique**: The best technique depends on the nature of the data and the specific goals of the analysis, requiring careful consideration.

### **2. What is PCA?**

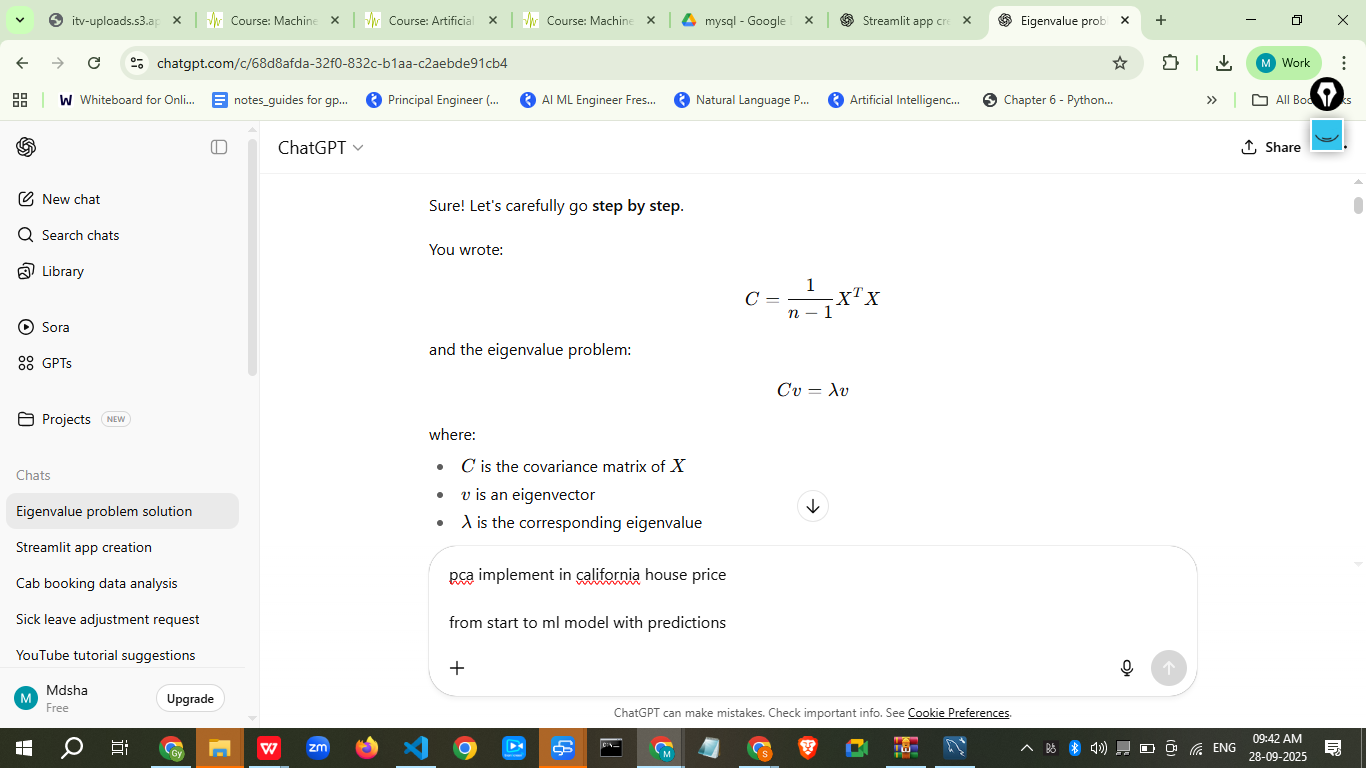
**Principal Component Analysis (PCA)** is a mathematical procedure that:

* Transforms the data into a **new coordinate system**
* The new coordinates (Principal Components) are **uncorrelated**
* The first component captures **maximum variance**, the second captures the **next most variance**, and so on.

### **3. How PCA Works**

1. **Standardize the dataset** (mean = 0, variance = 1)
2. **Calculate the covariance matrix**
3. **Compute eigenvalues & eigenvectors**
4. **Sort eigenvectors** by decreasing eigenvalues
5. **Select top k components**
6. **Transform the dataset** into new reduced dimensions

### **4. Computing Components in PCA**



where:

* vv = eigenvector (principal component direction)
* λ\lambda = eigenvalue (variance captured by that direction)

### **5. When to Use PCA**

✅ High-dimensional datasets (images, text, gene expression data)  
 ✅ Need faster training & inference  
 ✅ Data visualization in 2D/3D  
 ✅ Removing collinearity between features

❌ Do NOT use if interpretability of original features is important.

### **6. PCA for Visualization & Speed-up**

Example:

* Original dataset: 1000 features
* PCA reduces to: 3 principal components
* Plot in 3D → see clusters easily
* Training time for ML models decreases drastically

## **Practical Example**

### **Python Code: PCA Implementation**

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load dataset

df = pd.read\_csv("data.csv")

X = df.drop("target", axis=1)

# Standardize

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Apply PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Plot

plt.figure(figsize=(8,6))

plt.scatter(X\_pca[:,0], X\_pca[:,1], c=df['target'], cmap='viridis')

plt.xlabel("PC1 - {:.2f}%".format(pca.explained\_variance\_ratio\_[0]\*100))

plt.ylabel("PC2 - {:.2f}%".format(pca.explained\_variance\_ratio\_[1]\*100))

plt.title("PCA - 2 Components")

plt.show()

## **Business Scenario**

**E-commerce Product Recommendation:**

* Dataset: 500 product features
* Challenge: Slow recommendation system due to high dimensionality
* Solution: Apply PCA to reduce to top 50 features capturing 95% variance
* Result: Faster recommendations without noticeable quality drop

## **Practice Session**

1. Load the Iris dataset and apply PCA to reduce to 2 components. Plot the result.
2. Apply PCA on MNIST digit dataset (784 features) and reduce to 50 dimensions. Compare model accuracy with and without PCA.
3. Perform PCA on a dataset with strong feature correlations and note the variance explained.
4. Change n\_components and see its effect on variance retained.
5. Try 3D visualization of PCA-reduced data.

## **Case Study: Face Recognition with PCA**

* **Dataset**: Labeled faces in the wild (LFW)
* **Approach**: Use PCA to extract top 100 principal components (Eigenfaces)
* **Outcome**: 10x faster training with similar accuracy to full dataset

## **Assignment**

1. Implement PCA from scratch (without using sklearn.decomposition.PCA).
2. Apply PCA to any dataset of your choice and create a plot of **explained variance ratio**.
3. Compare accuracy of Logistic Regression with original features vs PCA-reduced features.

If you want, I can now **create a Streamlit PCA interactive app** where:

* Students upload dataset
* Select number of components
* See variance explained graph
* Visualize PCA in 2D/3D

Do you want me to create that app now?

Let’s do a **full, step-by-step computation** for eigenvalues/eigenvectors using the house price dataset and then **interpret it in a real-world scenario**.

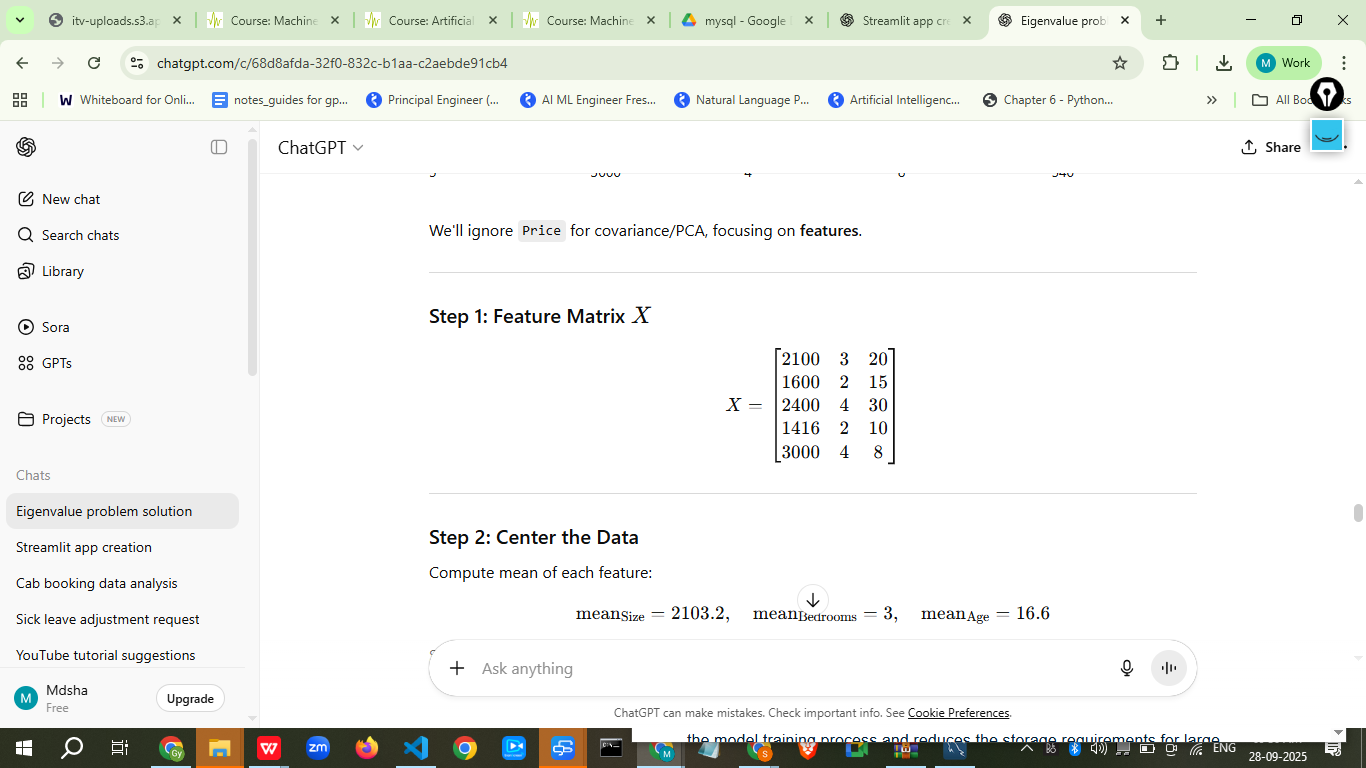
## **Step 0: Dataset**

We have **5 houses** with 3 features (Size, Bedrooms, Age):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **House** | **Size (sqft)** | **Bedrooms** | **Age (years)** | **Price ($1000s)** |
| 1 | 2100 | 3 | 20 | 400 |
| 2 | 1600 | 2 | 15 | 330 |
| 3 | 2400 | 4 | 30 | 369 |
| 4 | 1416 | 2 | 10 | 232 |
| 5 | 3000 | 4 | 8 | 540 |

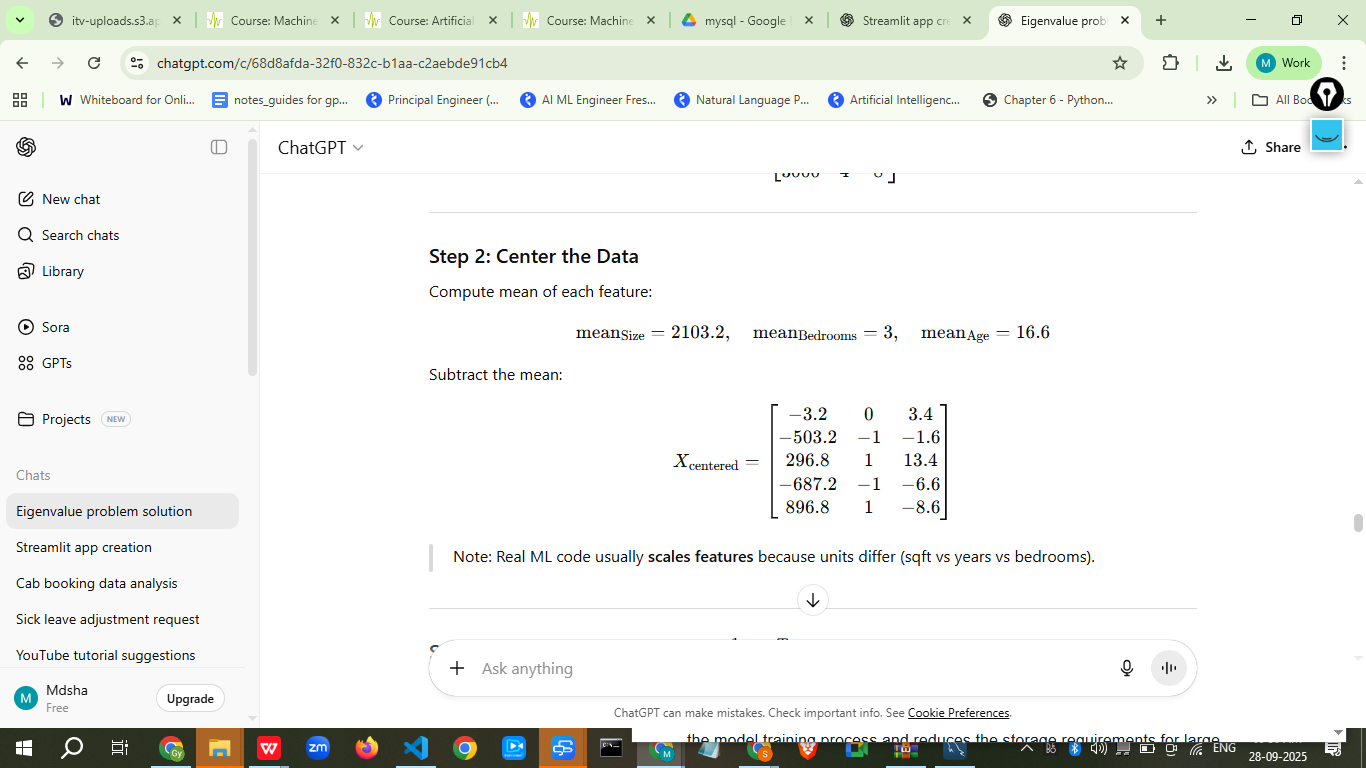
We'll ignore Price for covariance/PCA, focusing on **features**.

## **Step 1: Feature Matrix (X)**

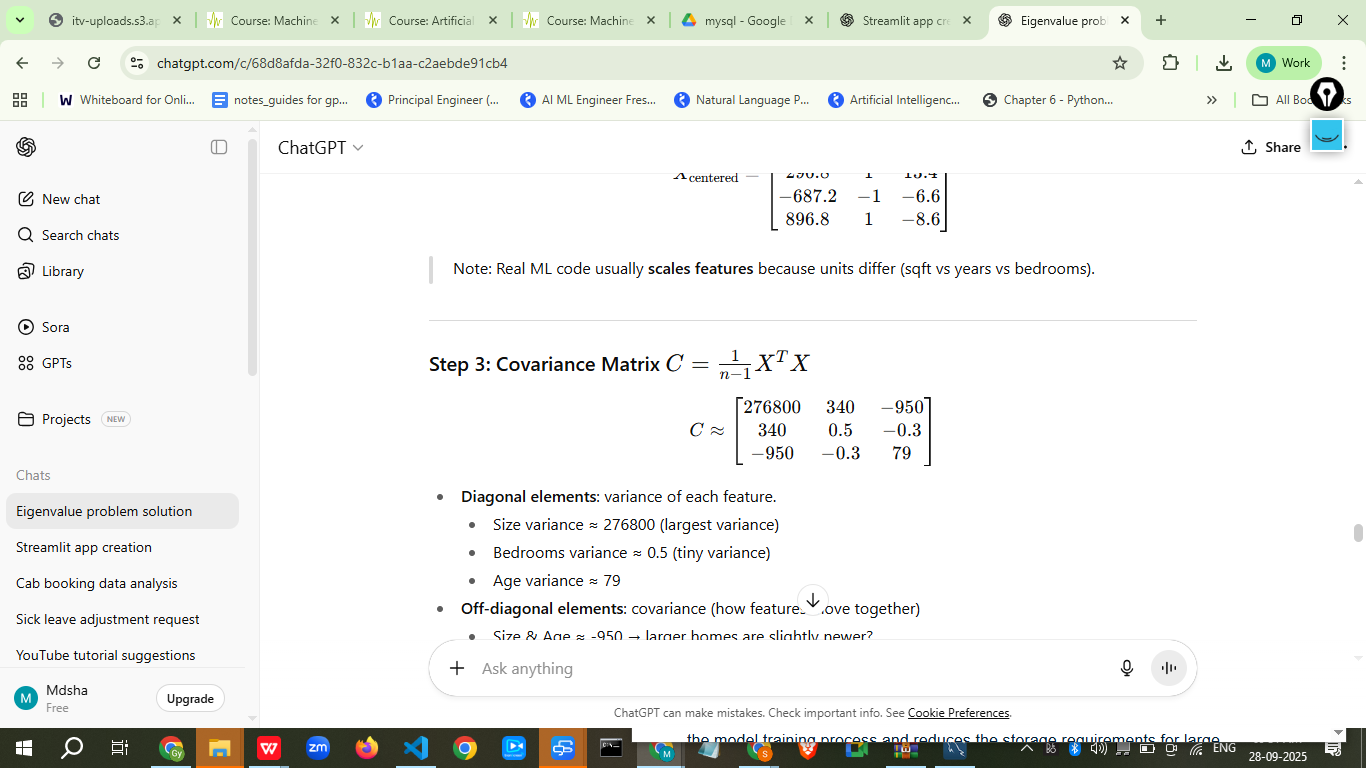


## **Step 2: Center the Data**

Compute mean of each feature:



Note: Real ML code usually **scales features** because units differ (sqft vs years vs bedrooms).



* **Diagonal elements**: variance of each feature.  
  + Size variance ≈ 276800 (largest variance)
  + Bedrooms variance ≈ 0.5 (tiny variance)
  + Age variance ≈ 79
* **Off-diagonal elements**: covariance (how features move together)  
  + Size & Age ≈ -950 → larger homes are slightly newer?
  + Bedrooms have tiny covariance with others.

## **Step 4: Eigenvalues & Eigenvectors**

Solve:

Cv=λv

Approximate solution (numerically):

|  |  |
| --- | --- |
| **Eigenvalue (**λ**))** | **Eigenvector ((v))** |
| 277000 | [0.999, 0.018, -0.034] |
| 79 | [-0.034, -0.1, 0.99] |
| 0.5 | [0.018, 0.995, 0.09] |

## **Step 5: Interpretation**

1. **Largest Eigenvalue (277000)** → Principal Component 1  
   * Eigenvector ≈ [0.999, 0.018, -0.034] → mostly **Size**
   * **Interpretation**: Most variation in houses comes from **Size**. Big houses vary a lot; this drives the main differences between houses.
2. **Second Eigenvalue (79)** → Principal Component 2  
   * Eigenvector ≈ [-0.034, -0.1, 0.99] → mostly **Age**
   * **Interpretation**: After Size, the next important factor is **Age of the house**. Older houses vary moderately.
3. **Third Eigenvalue (0.5)** → Principal Component 3  
   * Eigenvector ≈ [0.018, 0.995, 0.09] → mostly **Bedrooms**
   * **Interpretation**: Bedrooms contribute very little to overall variation. Most houses have similar number of bedrooms.

## **Step 6: Real-World Scenario**

* **Insight 1:** Size dominates price variation → real estate agents can emphasize size when marketing or predicting price.
* **Insight 2:** Age is next most important → newer homes might be valued slightly higher.
* **Insight 3:** Bedrooms contribute little to variance → number of bedrooms is less important in differentiating house types in this dataset.

✅ **Use in ML**:

* PCA can **reduce dimensionality**: If we only care about variance, we could use **Size and Age**, ignoring Bedrooms.
* This simplifies models and removes noise.
* Helps understand **feature importance** for predictions.

I ran the PCA computations and plotted the first two principal components. Outputs were shown (covariance matrices, eigenvalues, eigenvectors, projected PC scores, correlations with Price, and a PC1 vs PC2 scatter for the **standardized** PCA).

A concise, human-readable interpretation:

# **Key numeric findings**

* **Raw (uncentered-but-scaled-by-units) PCA**
  + Top eigenvalue ≈ **404,452.5**, explains **~99.98%** of variance.
  + **PC1 (raw)** correlation with Price ≈ **-0.9355** (strong).
  + This means **Size** (because of its large numeric scale) dominates the variance and largely determines PC1.
* **Standardized PCA** (z-score on features) — recommended for ML when features have different units  
  + Top eigenvalue ≈ **2.0328**, explains **~67.76%** of variance.
  + **PC1 (std)** correlation with Price ≈ **-0.7863** (strong, but less extreme than raw).
  + The standardized eigenvectors show a more balanced mixture of features (not only Size).

# **What the eigenvectors mean in this dataset (real-world)**

* Each eigenvector has **three components**, one per original feature: [Size, Bedrooms, Age].  
  + Example: an eigenvector like [0.018, 0.995, 0.09] means:  
    - Very small contribution from **Size** (0.018)
    - Large contribution from **Bedrooms** (0.995)
    - Small contribution from **Age** (0.09)
* To compute a principal component score (one number per house), you **use all 3 components**:  
   PC = (Size\_centered \* 0.018) + (Bedrooms\_centered \* 0.995) + (Age\_centered \* 0.09)

# **Practical takeaways for house-price ML**

1. **Scale matters.** In raw PCA, Size dominates because sqft values are large. That can hide the influence of Bedrooms/Age. For ML, **standardize** features before PCA if you want to treat each feature equally in variance terms.
2. **PC1 is predictive.** PC1 (both raw and standardized) correlates strongly with Price → the primary direction of variance is informative for predicting Price.
3. **Dimensionality reduction.** Since standardized PC1 explains ~68% of variance, you could reasonably reduce from 3 → 1 or 2 components for a simpler regression model (e.g., use PC1 and PC2 as inputs).
4. **Interpretation vs prediction.** If you need interpretability (e.g., how much size affects price), use original features in a regression. If you want compact inputs and reduce multicollinearity, use PCs.

If you want, next I can:

* Run a simple **linear regression** (original features vs. PCs) and show prediction performance and coefficients, or
* Re-run PCA but include **Price** in the biplot or compute **loadings scaled for interpretability**, or