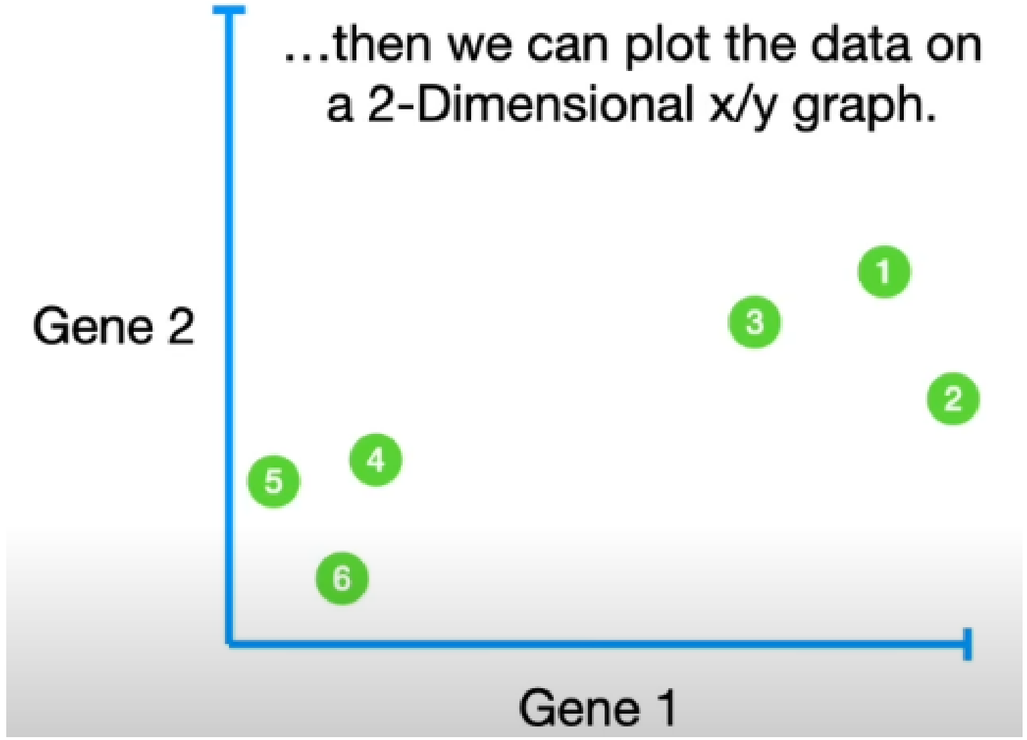
Principal Component Analysis(PCA)

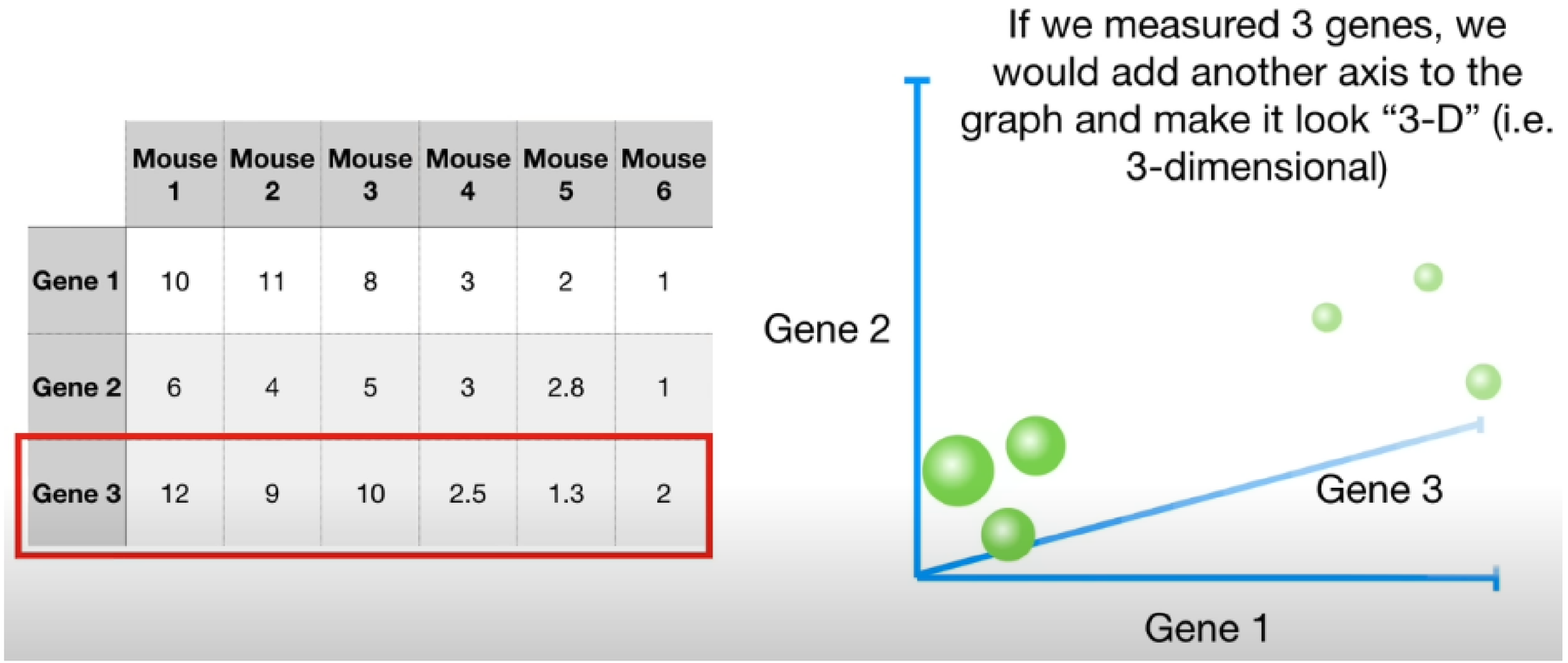
1) Intuitive way

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	1	2
Gene 2	6	4	5	3	2.8	1

Let's start with a simple data set



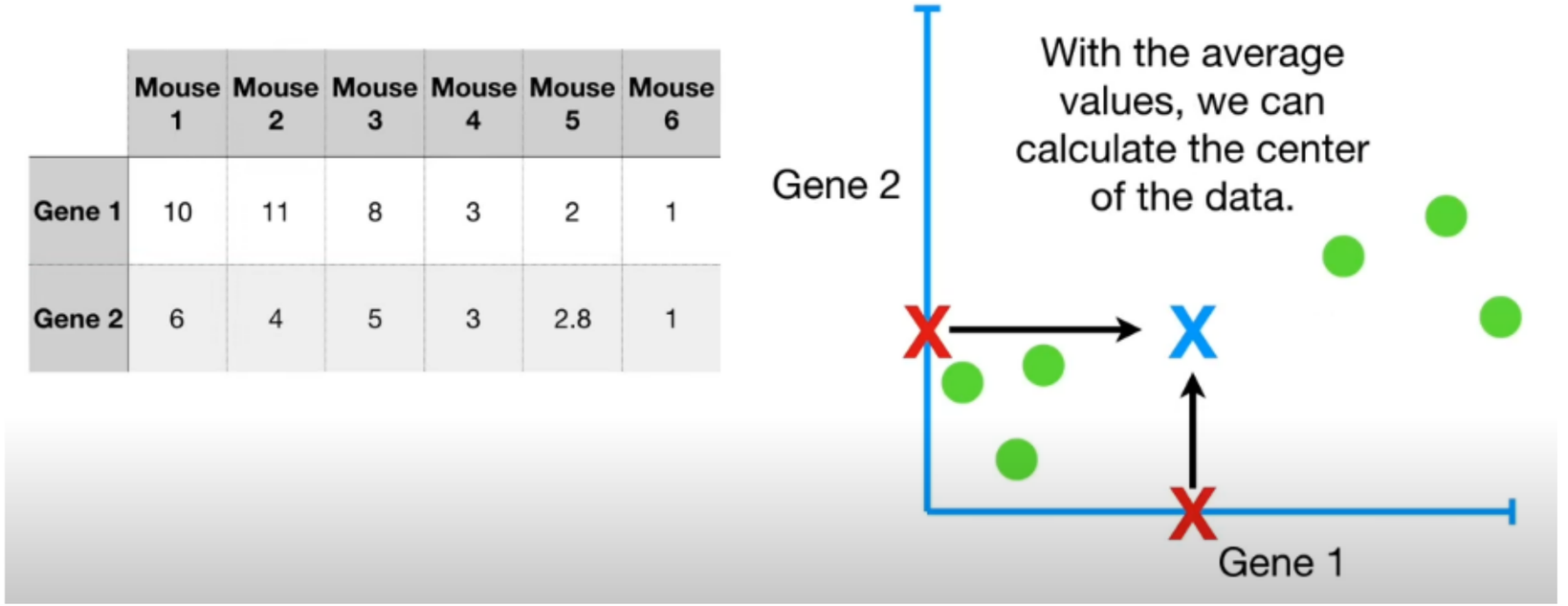
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1
Gene 3	12	9	10	2.5	1.3	2



In the similar manner, if we take 4 or more genes as features, we can no more plot them and see the clustering of the data points. So, we need a method by which we can reduce the dimension or reduce the number of features. We use PCA to do so.

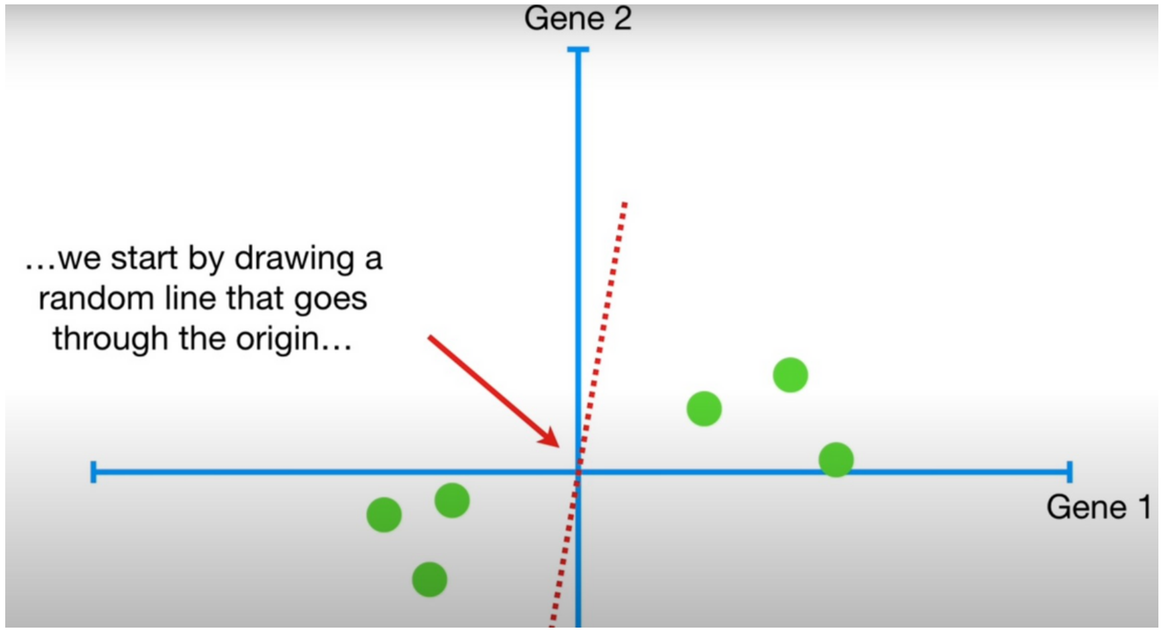
In case of a 2 dimensional data, we learn how to make a PCA plot and then we could similarily use PCA plots for higher dimensional datas

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1

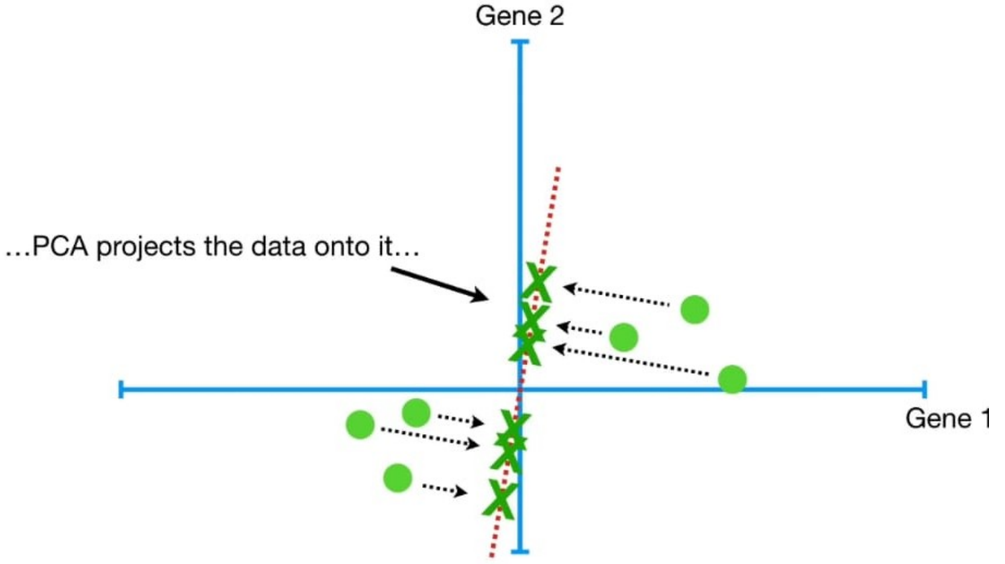


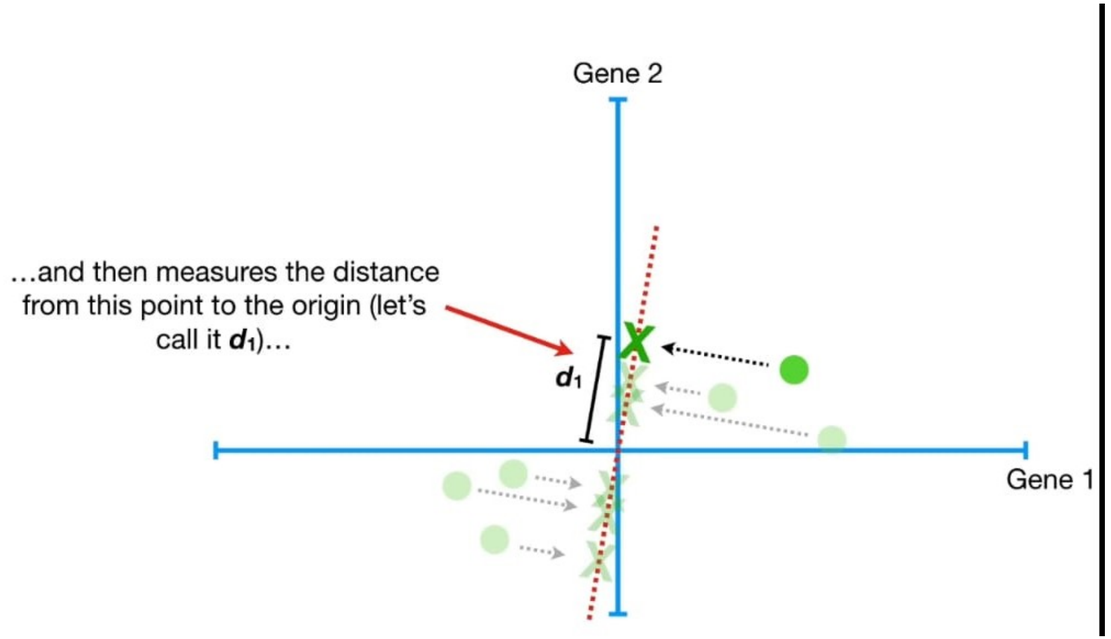
We calculate the centroid of these data points

Now we'll shift the data points so that the center is on top of the origin

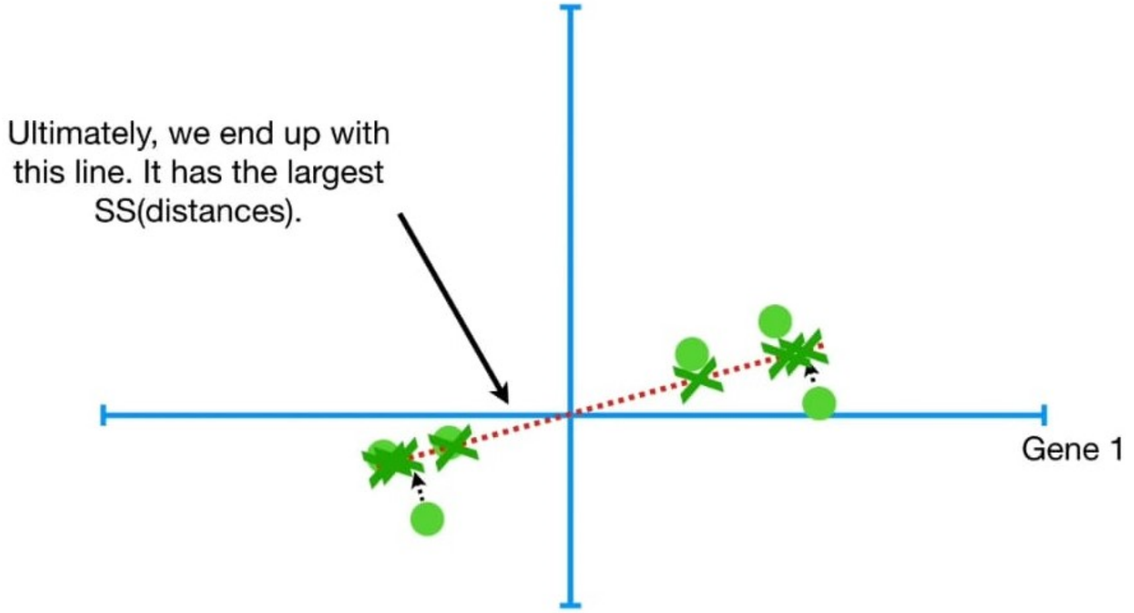


To find the optimized line, rotate the line about the origin in such a manner that the distances of the data from the line minimizes or the distance of their projections from origin maximizes





$d_1^2 + d_2^2 + d_3^2 + d_4^2 + d_5^2 + d_6^2$ = sum of squared distances = SS(distances)

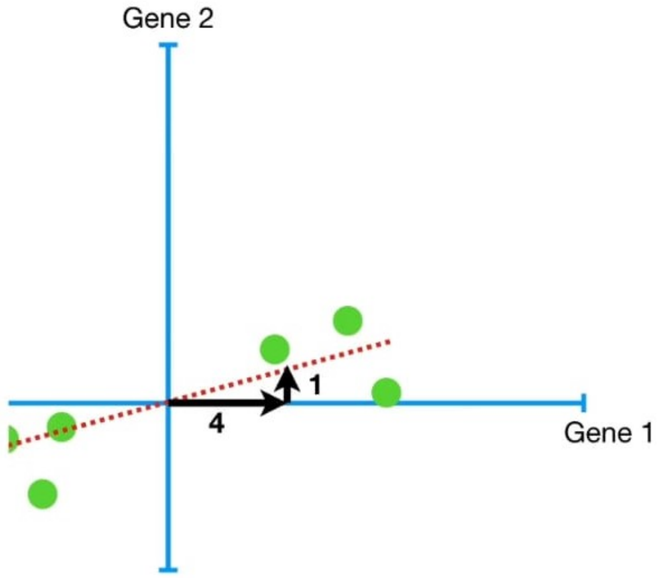


We need to maximize SS(distances) to find the optimized line. This optimized line is called PC1

Let's assume the slope of PC1 is 0.25

To make PC1
Mix **4** parts Gene 1
with **1** part Gene 2

I mention this because when someone says, “PC1 is a linear combination of variables…”

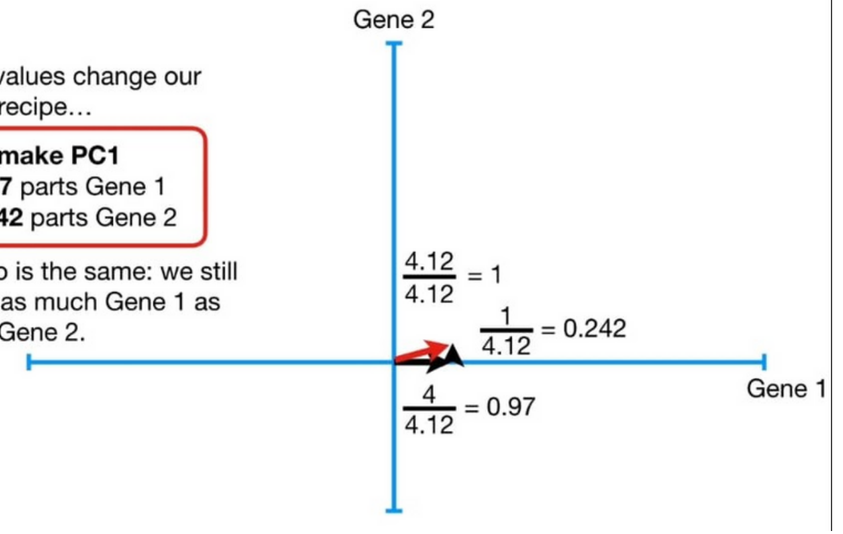


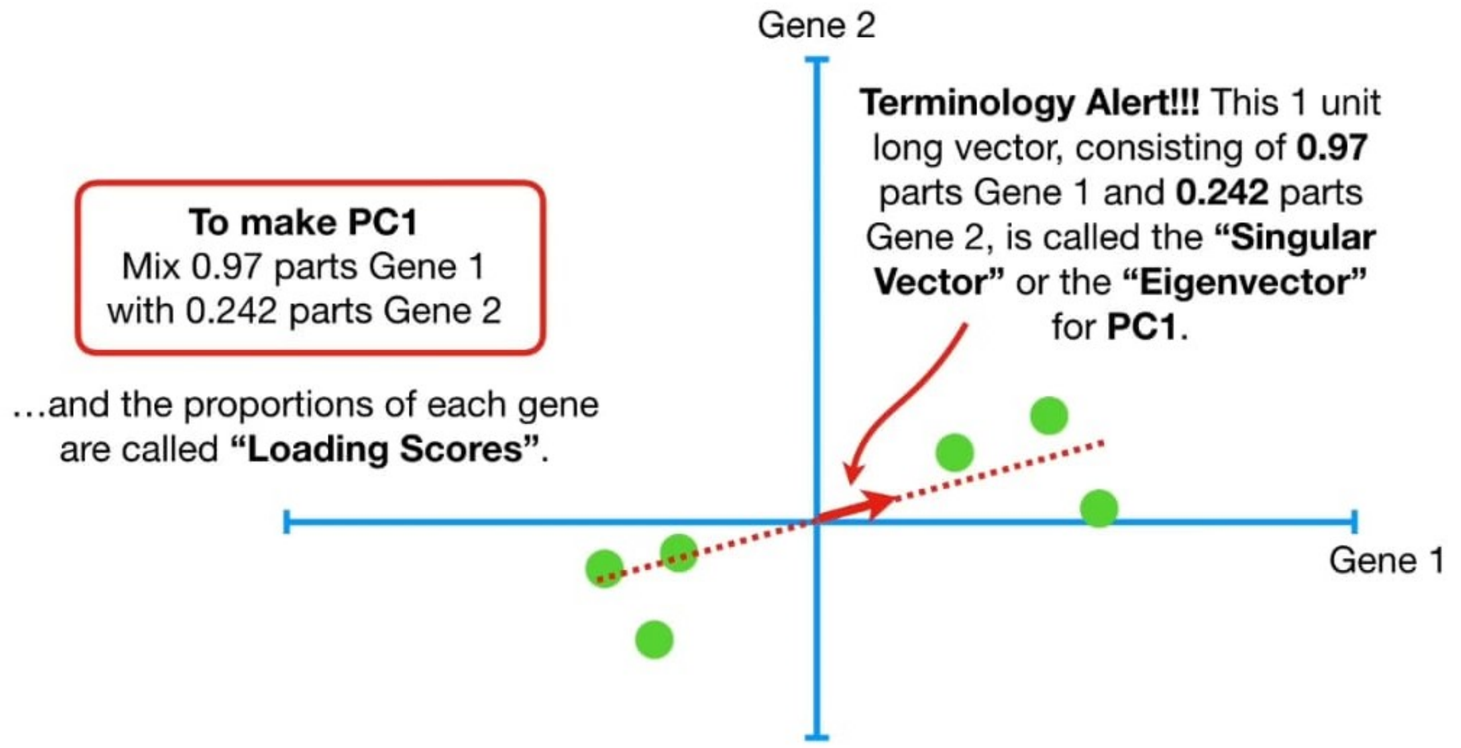
As the eigenvector is a unit vector, rescale the values. After rescaling them we get 0.97 parts of gene 1 and o.242 parts of gene 2

The new values change our recipe…

To make PC1
Mix **0.97** parts Gene 1
with **0.242** parts Gene 2

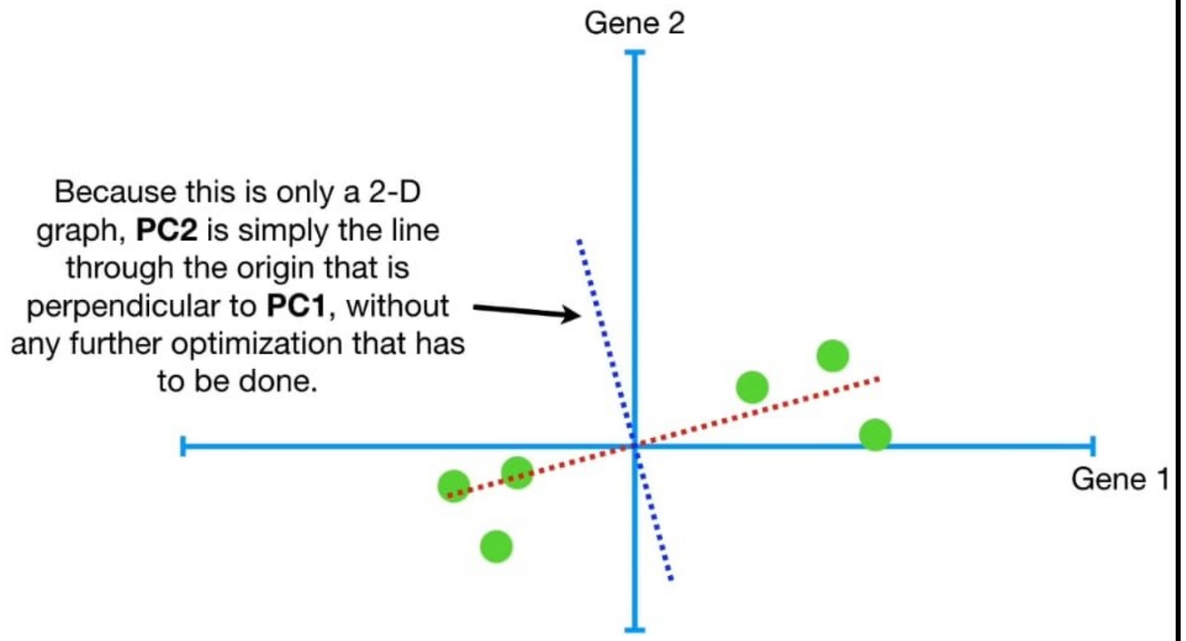
…but the ratio is the same: we still use 4 times as much Gene 1 as Gene 2.

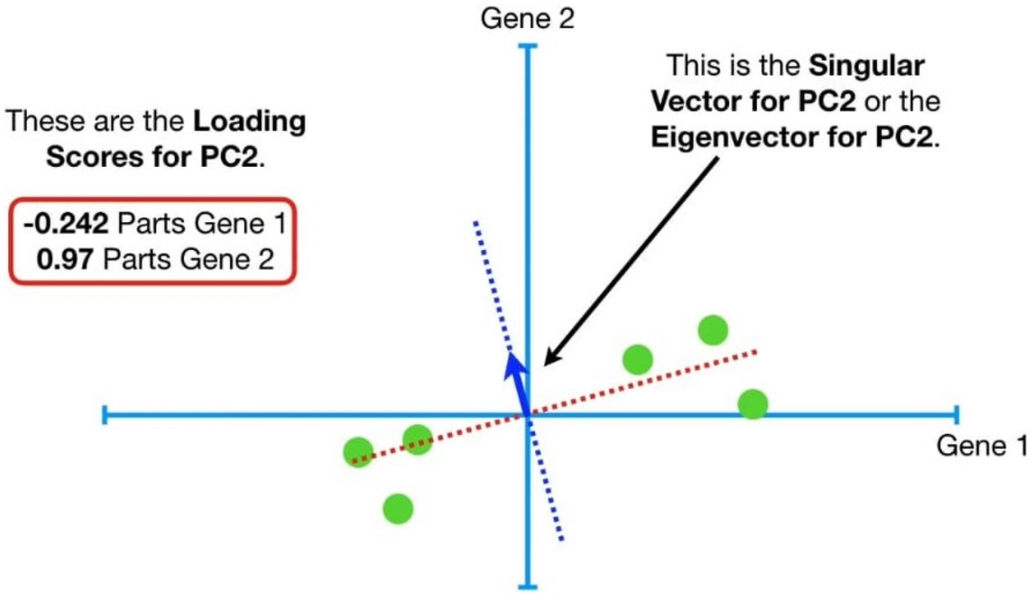




$$\frac{SS(\text{distances for PC1})}{n - 1} = \text{Eigenvalue for PC1}$$

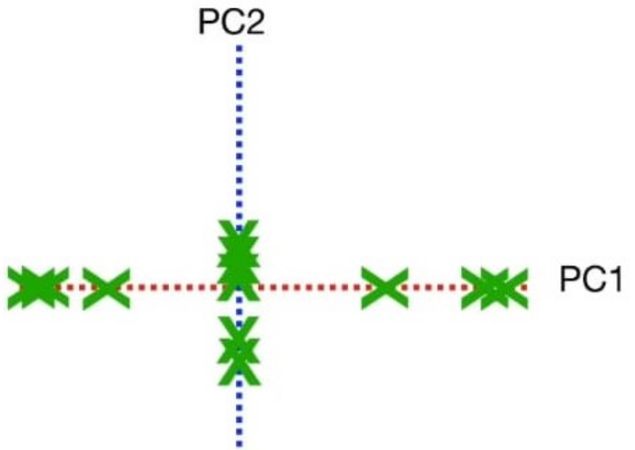
$$\sqrt{SS(\text{distances for PC1})} = \text{Singular Value for PC1}$$

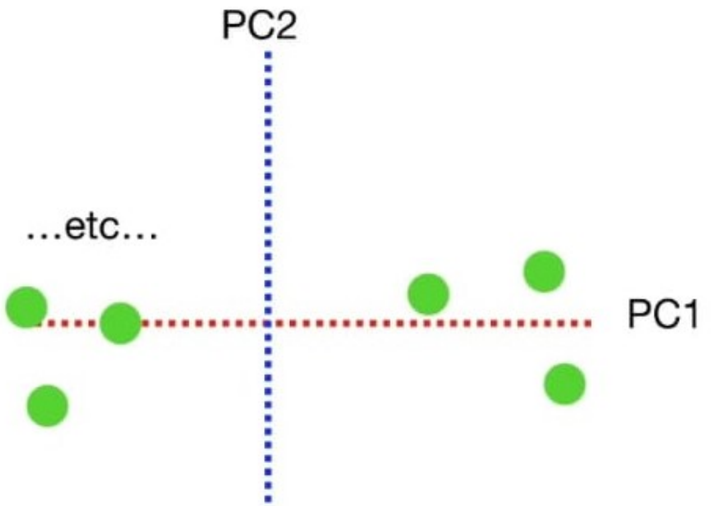




$$\frac{SS(\text{distances for PC2})}{n - 1} = \text{Eigenvalue for PC2}$$

…then we use the projected points to find where the samples go in the PCA plot.

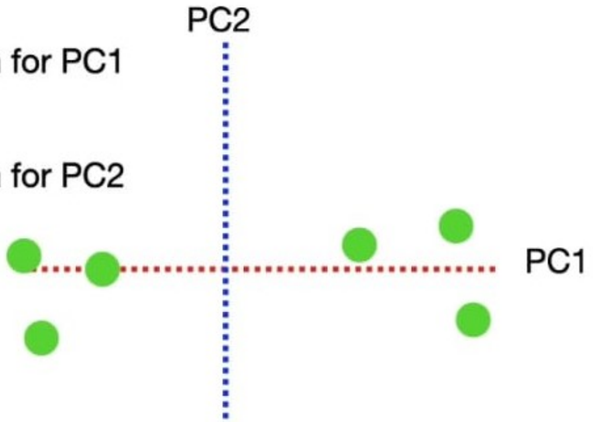




Well, if you are familiar with the equation for variation, you will notice **Eigenvalues** are just measures of variation.

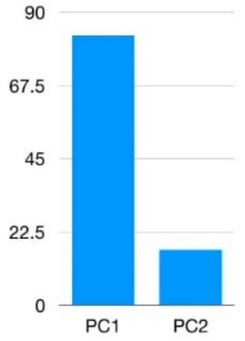
$$\frac{SS(\text{distances for PC1})}{n-1} = \text{Variation for PC1}$$

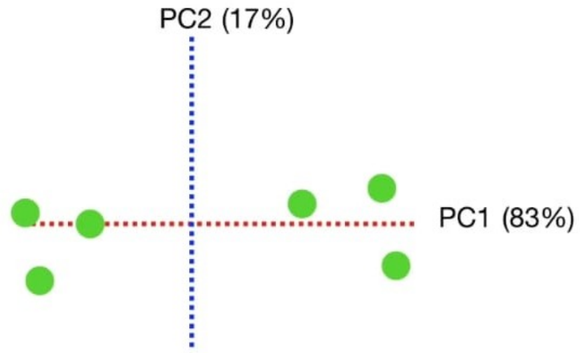
$$\frac{SS(\text{distances for PC2})}{n-1} = \text{Variation for PC2}$$



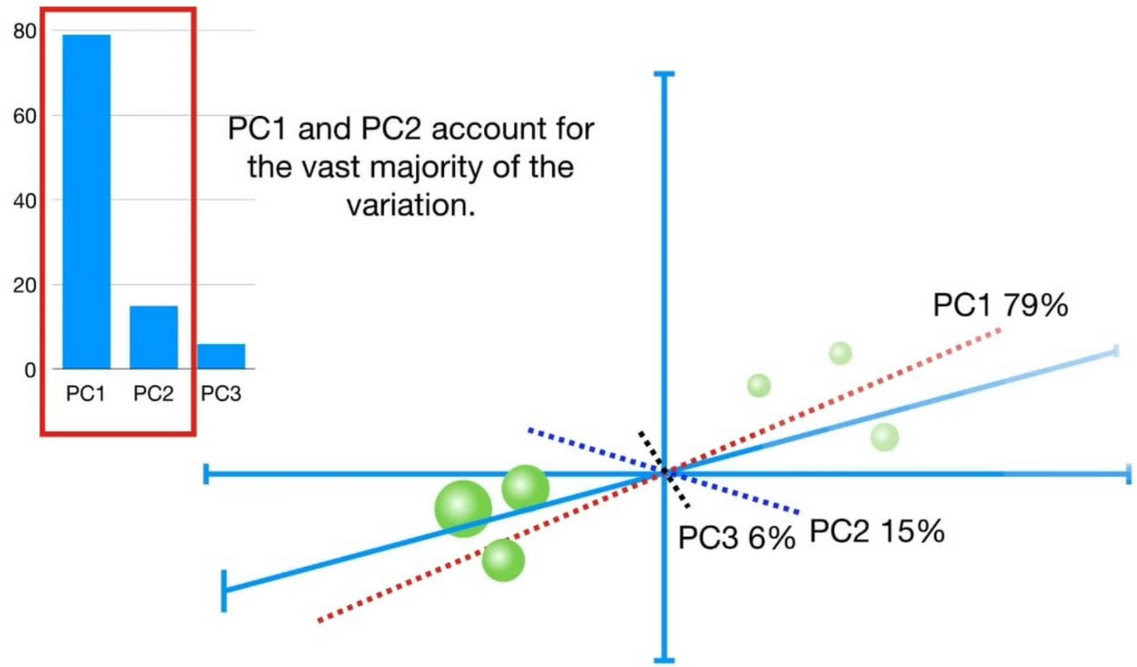
For example, the variance for PC1 is 15 and variance for PC2 is 3. This means total variation around both the PCs is 18. Therefore, PC1 has 83% of variation and PC2 has 17% of variation

TERMINOLOGY ALERT!!!! A **Scree Plot** is a graphical representation of the percentages of variation that each PC accounts for.



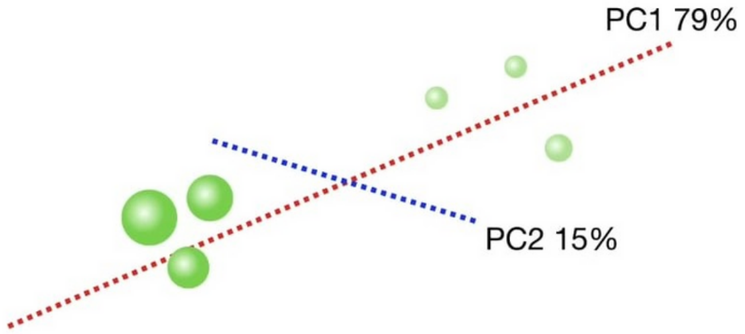


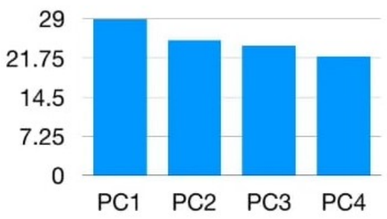
Let's take an example of a 3-dimensional data in which variations of PC1,PC2, and PC3 are 79%,15% and 6% respectively -



In order to reduce the dimension, we ignore PC3 as it's variation is only 6% so it can be ignored. Rest 94% is already covered by PC1 AND PC2.

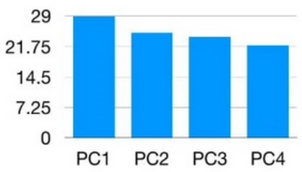
To convert the 3-D graph into a 2-D PCA graph, we just strip away everything but the data and PC1 and PC2...

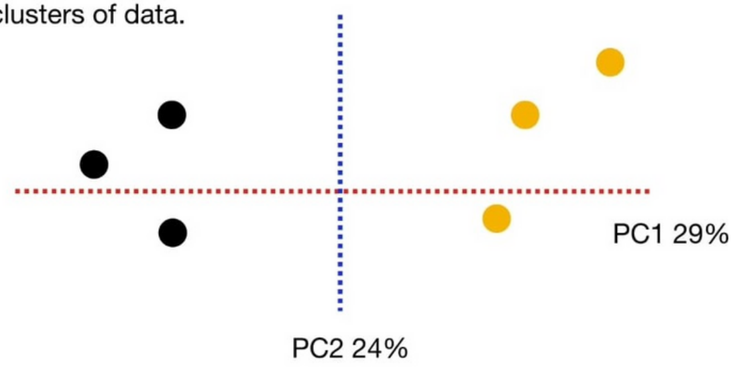




NOTE: If the scree plot looked like this, where PC3 and PC4 account for a substantial amount of variation, then just using the first 2 PCs would not create a very accurate representation of the data.

However, even a noisy PCA plot like this can be used to identify clusters of data.





Few things to keep in mind:-

1) Scaling your data
2) Centering your data
3) How many principal components you should expect to get.

2) Mathematical way

Standard deviation and variance only operate on 1 dimension, so that you could only calculate the standard deviation for each dimension of the data set independently of the other dimensions. However, it is useful to have a similar measure to find out how much the dimensions vary from the mean with respect to each other.
Covariance is such a measure. Covariance is always measured between 2 dimensions. If you calculate the covariance between one dimension and itself, you get the variance.
The formula for variance could also be written like this: $$var(X) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

Expand the square term to two terms : $$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

When there are more than two dimensions, there would more number of covariance relations, to include them all, we introduce covariance matrix.

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Procedure

1. Take the matrix of independent variables Z.
2. For each column, subtract the mean of that column from each entry. (This ensures that each column has a mean of zero.)
3. Take the matrix, transpose it, and multiply the transposed matrix with the matrix itself. (Z^T Z.) The resulting matrix is the covariance matrix of Z, up to a constant.
4. Find the eigenvectors and the eigenvalues of the covariance matrix.
5. Sort the eigenvectors in the order of decreasing eigenvalues. Make a matrix using the first k eigenvectors, call it P* .
6. Calculate Z* = Z P*

https://colab.research.google.com/drive/1Fy56oCLDnOWBDKdlUj5Wuujs7-xFIwZg?usp=sharing#scrollTo=eMlFrSICE5zW

```python
def pca(X, n_components):
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean
    cov_matrix = np.cov(X, rowvar=False)
    eig_values, eig_vectors = np.linalg.eig(cov_matrix)
    sorted_indices = np.argsort(eig_values)[::-1]
    sorted_eig_vectors = eig_vectors[:, sorted_indices]
    top_eig_vectors = sorted_eig_vectors[:, :n_components]
    X_pca = np.dot(X_centered, top_eig_vectors)
    return X_pca
```