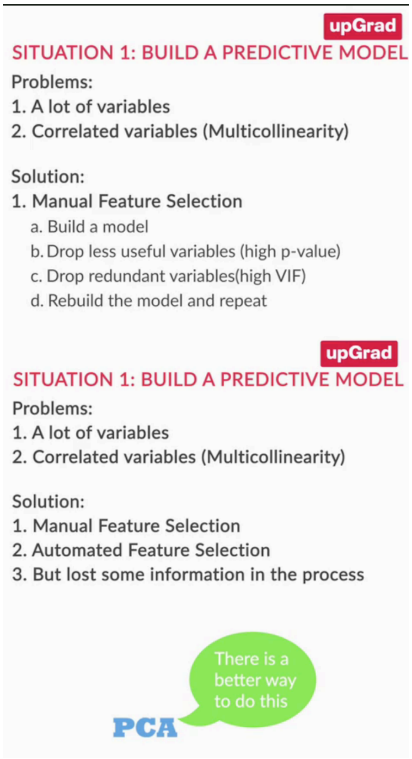
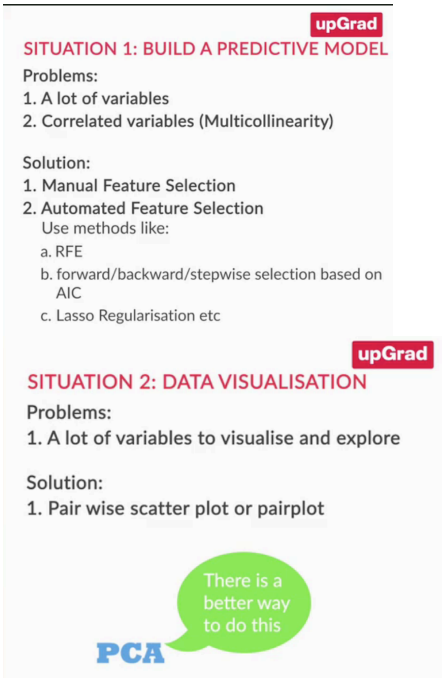
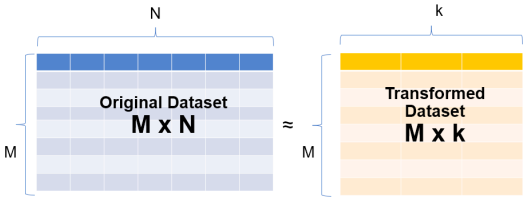
Principle Component Analysis (PCA):

Principal component analysis (PCA) is one of the most commonly used dimensionality reduction techniques in the industry. By converting large data sets into smaller ones containing fewer variables, it helps in improving model performance, visualising complex data sets, and in many more areas.
Why PCA: here are some situations.





PCA is a dimensionality reduction technique, i.e., it approximates the original data set to a smaller one containing fewer dimensions(Note that ***dimension is just another term for referring to columns or variables in a dataset***)



PCA helps

- For data visualisation and EDA
- For creating uncorrelated features that can be input to a prediction model: With a smaller number of uncorrelated features, the modelling process is faster and more stable as well.
- Finding latent themes in the data: If you have a data set containing the ratings given to different movies by Netflix users, PCA would be able to find latent themes like genre and, consequently, the ratings that users give to a particular genre.
- Noise reduction

what PCA does is that it converts the data **by creating new features from old ones**, where it becomes easier to decide which features to consider and which not to.

PCA is a statistical procedure to convert observations of possibly correlated variables to 'principal components' such that:

- They are **uncorrelated** with each other.
- They are **linear combinations** of the original variables.
- They help in capturing maximum **information** in the data set.
- **PCA is an unsupervised technique!**

Now, the aforementioned definition introduces some new terms, such as '**linear combinations**' and '**capturing maximum information**', for which you will need some knowledge of linear algebra concepts as well as other building blocks of PCA. In the next session, we will start our journey in the same direction with the introduction of a very basic idea: the **vectorial representation of data**.

Advantages of PCA:

Being able to eliminate unwanted features without major information loss.

Disadvantages:

the model wont use the original features any more and the pcs wont be interpretable. As in industry interpretability is top priority.

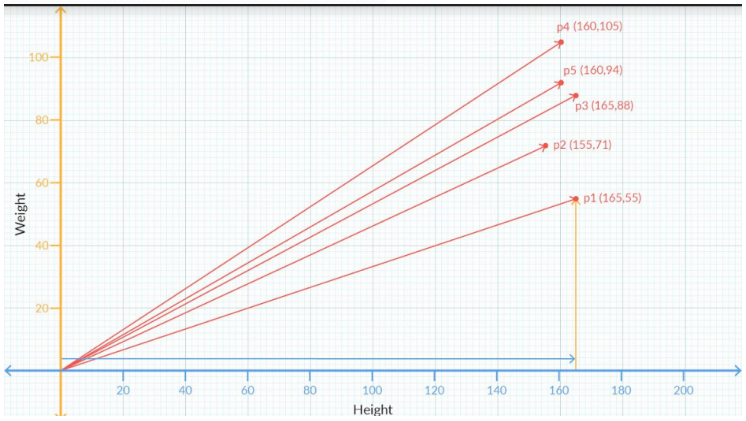
Vectorial representation of data:

Let us consider an example of the following data:

Patient ID	Height (cm)	Weight (kg)
P1	165	55
P2	155	71
P3	165	88
P4	160	105
P5	160	94

The height and weight info can be represented in the form of a matrix as shown in the adjacent figure. These patient details can be individually represented as showing in the graph below:

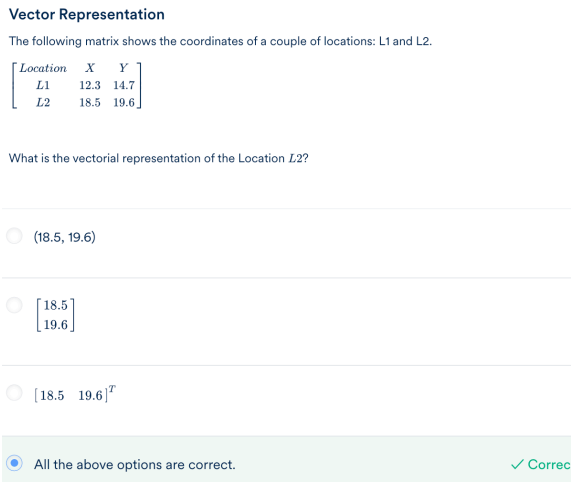
$$\begin{bmatrix} 165 & 55 \\ 155 & 71 \\ 165 & 88 \\ 160 & 105 \\ 160 & 94 \end{bmatrix}$$



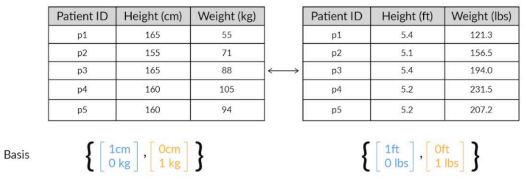
Vector Representation

The vector associated with the first patient is given by the values (165, 55). This value can also be written in the following way:

1. 1. A column containing the values along the rows. This is also known as the column-vector representation. [165
   55]
2. As a transpose of the above form. Essentially, it is the same column vector but now written as a transpose of a row vector. [165 55]^T
   [Note: Transpose is something you must have learnt in your Python for DS module. If you need some brushing up on this topic, you can take a look at this link]
3. In terms of the basis vectors
   This is something which you'll learn in detail in later segments. To give a brief idea, the vector (165,55) can also be written as 165**i** +55**j**, where **i** and **j** are the unit vectors along X and Y respectively and are the basis vectors used to represent all vectors in the 2-D space.



The fundamental role of PCA is to change the basis in such a way as to represent the data in as normal way as reduced dimensions as possible.



We learnt the following formula

New Basis Representation = M x Old Basis Representation

M is the representation of the old basis in the new basis

It transforms a point's representation from the old basis to the representation in the new basis

Next, we'll see how we can derive M when we're given the old and the new basis vectors.

- Rearranging the previous equation once again

$$\begin{bmatrix} 1\ ft & 0 \\ 0 & 1\ lbs \end{bmatrix} \begin{bmatrix} 5.4 \\ 121.3 \end{bmatrix} = \begin{bmatrix} 1\ cm & 0 \\ 0 & 1\ kg \end{bmatrix} \begin{bmatrix} 165 \\ 55 \end{bmatrix}$$

- Keeping everything in the same coordinate system

$$\begin{bmatrix} 1\ ft & 0 \\ 0 & 1\ lbs \end{bmatrix} \begin{bmatrix} 5.4 \\ 121.3 \end{bmatrix} = \begin{bmatrix} 0.0328\ ft & 0 \\ 0 & 2.205\ lbs \end{bmatrix} \begin{bmatrix} 165 \\ 55 \end{bmatrix}$$

- Now from the new equation that we had derived

$$B_1 * v_1 = B_2 * v_2$$

Let's do a simple manipulation $B_2^{-1} * B_1 * v_1 = B_2^{-1} * B_2 * v_2$

This gives us $v_2 = B_2^{-1} * B_1 * v_1$

Comparing this result with the equation from the previous slide $M = B_2^{-1} * B_1$

- Here's the equation once again

$$B_1 * v_1 = B_2 * v_2$$

- B_1 represents one set of basis vectors and v_1 is a point's representation in B_1
- B_2 represents a different set of basis vectors and v_2 is the same point's representation in B_2
- Using this equation we'll derive the value of M.

The final equation, after removing the units

$$\underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{B_1} \underbrace{\begin{bmatrix} 5.4 \\ 121.3 \end{bmatrix}}_{v_1} = \underbrace{\begin{bmatrix} 0.0328 & 0 \\ 0 & 2.205 \end{bmatrix}}_{B_2} \underbrace{\begin{bmatrix} 165 \\ 55 \end{bmatrix}}_{v_2}$$

Or in other words

$$B_1 * v_1 = B_2 * v_2$$

- Here's a short recap
  - First, we derived the following relationship when we move between 2 different basis vectors $B_1 * v_1 = B_2 * v_2$
  - Next, we used a set of conventions to denote the movement from one basis to another. This gave us the following equation for M $v_2 = M * v_1$
  - From the first equation we derived the following relationship $v_2 = B_2^{-1} * B_1 * v_1$
  - Combining the 2nd and the 3rd equations we obtained the equation $M = B_2^{-1} * B_1$

Shortcut for finding M

Once we've correctly set the conventions and when our movement is between standard and non-standard basis vectors, it's possible to express M in terms of just the non-standard basis vectors

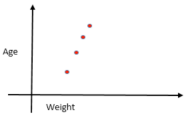
- If we're moving from a standard basis to a non-standard basis, M is equal to the inverse of the non-standard basis vectors
- If we're moving from a non-standard basis to a standard basis, M is equal to the non-standard basis vectors.

Standard basis = [[1,0],[0,1]]

Nonstandard basis = basis into which the the system is changing into.

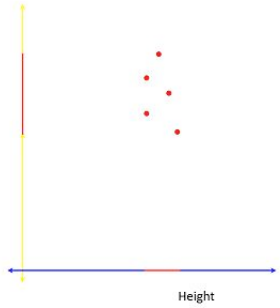
PCA: Variance

Which axis captures more variance in the following plot between Age and Weight?



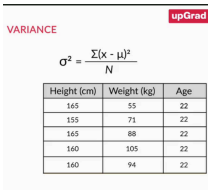
Age ✓ Correct

Feedback:
When the farthest points of the data are projected on the axis, the length of the projection becomes proportional to the variance. In the given image, the length of the projection of the farthest points of data on the Y-axis is more than the length of the projection on the X-axis.



Variance:

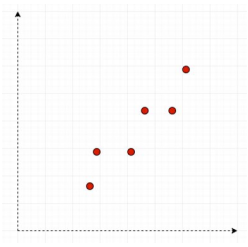
what PCA does. It changes the basis vectors in such a way that the new basis vectors capture the maximum variance or information
Basically, the steps of PCA for finding the principal components can be summarised as follows.



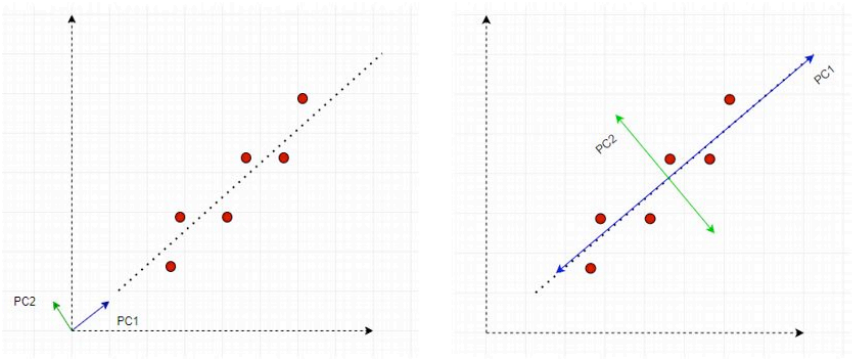
Height (cm)	Weight (kg)	Age
165	55	22
155	71	22
165	88	22
160	105	22
160	94	22

- First, it finds the basis vector which is along the best- fit line that maximises the variance. This is our first **principal component or PC1.**
- The second principal component is perpendicular to the first principal component and contains the next highest amount of variance in the dataset.
- This process continues iteratively, i.e. each new principal component is perpendicular to all the previous principal components and should explain the next highest amount of variance.
- If the dataset contains ***n*** independent features, then PCA will create ***n*** Principal components.

For a 2-D dataset that has the representation as shown in the image below:



the principal components can be visually represented as shown in the image below (click on the image to enlarge it):



Also, once the Principal Components are found out, PCA assigns a %age variance to each PC. Essentially it's the fraction of the total variance of the dataset explained by a particular PC. This helps in understanding which Principal Component is more important than the other and by how much. This is shown in the images below

Point	X	Y
P1	2	1
P2	3	1.5
P3	4	2
P4	6	3
P5	7	3.5
P6	8	4
Variance	**4.67**	**1.17**
%age of variance or information	**4.67/(4.67+1.17) = 80%**	**1.17/(1.17+4.67) = 20%**

Point	Xnew(PC1)	Ynew(PC2)
P1	2.24	0
P2	3.35	0
P3	4.47	0
P4	6.71	0
P5	7.83	0
P6	8.94	0
Variance	**5.83**	**0**
%age of variance or information	**5.83/(5.83 + 0) = 100%**	**0/(0+5.83) = 0 %**

Since 100% of the total variance or information of the entire dataset is present in only one of the columns (PC1) we can safely drop PC2 and still be assured of losing no information.

Before we end this session...

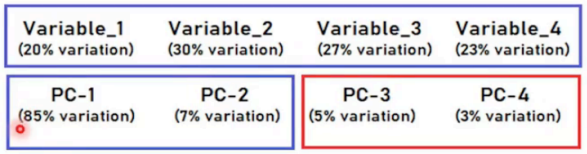
- The methodology or the algorithm by which PCA maximises the variance and obtains the new basis vectors is the process of ***eigendecomposition of the covariance matrix.***
- Using the eigendecomposition method, you'll be able to obtain the new basis vectors that will function as the Principal Components numerically. These new basis vectors are also called ***eigenvectors.***
- For example, in the roadmap case, the following PCs are obtained using the eigendecomposition of the covariance matrix of the original dataset.

$$\begin{bmatrix} 0.8944 \\ 0.4472 \end{bmatrix} \quad \begin{bmatrix} -0.4472 \\ 0.8944 \end{bmatrix}$$

Note - The number of principal components is the same as the number of columns in the dataset. PCs are sorted in descending order of information content.

PCA acts as a pre – processing tool in the ML pipeline, predominantly used for dimensionality reduction to improve model performance.

The approach is as follows



Note - The number of principal components is the same as the number of columns in the dataset. PCs are sorted in descending order of information content.

From the above picture, even if we choose to leave PC-3 and PC-4, we'll still retain 92% of the total information for only two dimensions of data which in turn will speed up the machine learning model.
Covariance Matrix: In order to capture the covariance or the correlations between the columns, we use the covariance matrix.
We are looking at variance of a column itself and its covariance with other columns.

The covariance between 2 columns X and Y is given by the following formula

$$cov(X,Y) = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{N}$$

The values may not be calculated manually as there is a predefined function in python already available to calculate the covariances.

Here's a list of properties of the covariance matrix:

1. For a matrix having N columns, the covariance matrix is an $N\ X\ N$ matrix.
2. If the columns are denoted by $X_1, X_2, X_3, X_4 \ldots$ and so on then the element in the i_{th} row and j_{th} column of the covariance matrix is given by the covariance between the columns X_i and X_j or $cov(X_i, X_j)$.
3. If $i = j$ then $cov(X_i, X_j)$ = $cov(X_i, X_i)$ essentially denotes the variance of the X_i column.
4. For a $3\ X\ 3$ matrix with columns **x**,**y** and **z**, the covariance matrix is given by the following matrix.

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

5. Also, as you can notice $cov(x, y) = cov(y, x)$ since both of them explain the covariance between x and y columns. In general, $cov(X_i, X_j) = cov(X_j, X_i)$

You have already learnt about the concept of **transformation matrix** in the previous segments. Let's try to recollect that learning here. When a given matrix is multiplied to a vector, then it simply transforms that particular vector to a new vector.

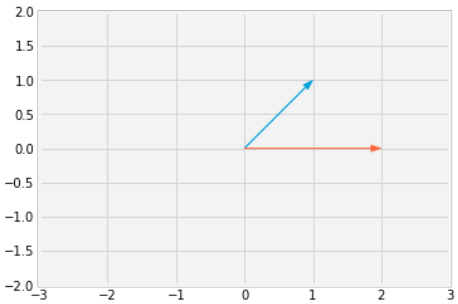
Now, as you saw in the video above, there is a vector 'x', which is represented as:

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

When you multiply this vector 'x' with matrix 'A', you get the following new transformed vector 'Ax':

where $A = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ and $Ax = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

As A is not a diagonal matrix, it'll result in change in both magnitude and direction. You can visualise these two vectors 'x' and 'Ax' graphically as shown below; the given vector 'x' is shown in blue and the transformed vector 'Ax' is shown in red.



Let's assume a transformation matrix 'A', which is shown below. $A = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$

In the x–y plane, the unit magnitude vectors in the direction of the x and y axes are simply denoted by 'i' and 'j', respectively, which can be represented in matrix form as shown below:

$$\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } \hat{j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Now, suppose you have two vectors 'x' and '^i', as shown below: $x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ and $\hat{i} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

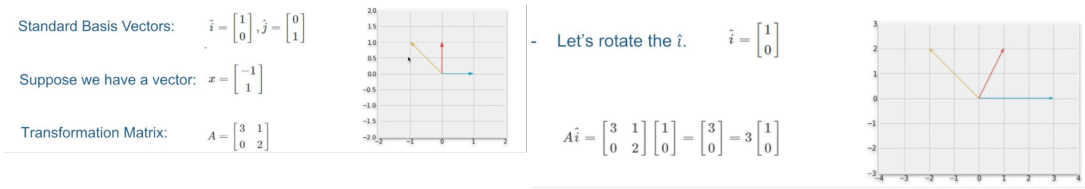
Let's transform these two vectors using the transformation matrix 'A' and examine what the new transformed vectors would be in this case:

$$Ax = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix} = 2 \begin{bmatrix} -1 \\ 1 \end{bmatrix} \text{ and } A\hat{i} = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

An interesting point that you should note here is that when you transform the 'x' and '^i' vectors using the matrix 'A', you get new vectors that are simply the scaled vectors of the original vectors, or, in other words, the transformed vectors are parallel to the original vectors. At the same time when you transform the 'x' and '^i' vectors using the transformed matrix 'A', you get the vectors, '2x' and '3^i' vectors, respectively.
So, in this case, multiplying the vectors 'x' and '^i' with the matrix 'A' has the same effect as simply multiplying them by the scalars 2 and 3 respectively.

Hence, the vectors 'x' and '^i' are called the eigenvectors of matrix 'A', and the scalars '2' and '3', respectively, by which these vectors get scaled are called the eigenvalues of matrix 'A'.

Eigen vector: an eigenvector of a linear transformation (or a square matrix) is a non-zero vector that changes at most by a scalar factor when that linear transformation is applied to it. The corresponding eigenvalue is the factor by which the eigenvector is scaled.



$$Ax = \lambda x$$

Multiply x with a matrix A has the same effect as multiplying it with a scalar.

The scalar is called as *eigenvalue* of A.

$$\lambda = eigenvalue$$

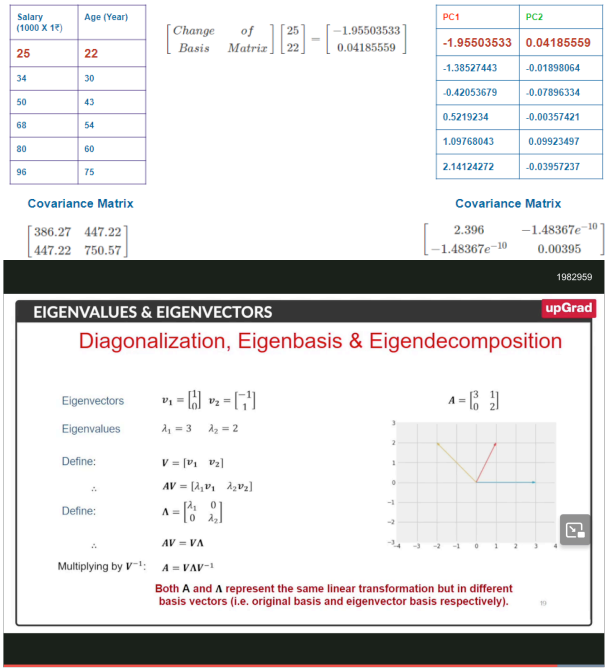
the eigenvectors (v1 and v2) of a 2X2 matrix ‘M’ that rotates any vector by an angle of 90 degree counterclockwise are imaginary Eigen vectors. Can be found using the following lines of code

```
A = [[0,-1],[1,0]]
np.linalg.eig(A)
```

The number of Eigen values and Eigen vectors of a matrix are same as the order of the matrix.
So, let’s list down some of the interesting properties of eigenvectors:

- Eigenvalues and eigenvectors of a given matrix always occur in pairs
- Eigenvalues and eigenvectors are defined only for square matrices, and it is not necessary that they will always exist. This means there could be cases where there are no eigenvectors and eigenvalues for a given matrix, or, in other words, there exist imaginary eigenvectors and eigenvalues.

Diagonalization of Covariance matrix:



Therefore, we can say that both A and Λ represent the same linear transformation but in different basis vectors (i.e., original basis and eigenvector basis, respectively).

Or, in other words, you can diagonalise matrix 'A' by representing it in a new eigenvector basis system, because matrix 'A' will become 'Λ' in the eigenvector basis system.

$$A = V\Lambda V^{-1}$$ Non-Diagonal Covariance matrix (A) in original space.

$$\Lambda = V^{-1}AV$$ Diagonalized Covariance matrix (Λ) in eigenvector space.

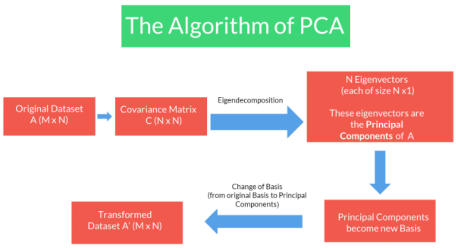
$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} = \Lambda$$

NOTE: number of principle components (K)<= no. of variables (n)

Let's summarise the steps of PCA algorithm one-by-one:

- Suppose you have an original data set with **'M' rows and 'N' columns**. Then you will have a **covariance matrix of order 'N X N'** for this data set.
- After getting the covariance matrix for the data set, find its eigenvectors, which will be the new basis vectors.
- Once you have obtained the eigenvectors, simply arrange them in the form of a matrix. This eigenvector matrix will be the **'change of basis matrix'**.
- After getting the **'change of basis matrix'**, multiply each data point in the original basis system with the inverse of **'change of basis matrix'** to get the new data points in the **eigenvector basis system.**
- In this way, you represent all the data points of the original basis system in the eigenvector basis system so that you can obtain the uncorrelated features as the covariance matrix has now been diagonalised.

One point that you should remember is that the **'eigenvectors of the covariance matrix are the new directions in which you represent the data in order to get uncorrelated features; or, in other words, the eigenvectors are the new principal components'.**



Model building using PCA:
Some important problems with this process that Rahim pointed out are:

- **Multicollinearity** among a large number of variables, which is not totally avoided even after reducing variables using RFE (or a similar technique)
- Need to use a **lengthy iterative procedure**, i.e. identifying collinear variables, using variable selection techniques, dropping insignificant ones etc.
- A **potential loss of information** due to dropping variables
- **Model instability** due to multicollinearity

Model Building with PCA

In the second part, first, we'll reduce the dimensions that we have using PCA and then create a logistic regression model on it.

As you could see, with PCA, you could achieve the same results with just a couple of lines of code. It will be helpful to note that the baseline PCA model has performed at par with the best Logistic Regression model built after the feature elimination and other steps.

PCA helped us solve the problem of multicollinearity (and thus model instability), loss of information due to the dropping of variables, and we don't need to use iterative feature selection procedures. Also, our model becomes much faster because it has to run on a smaller dataset. And even then, our ROC score, which is a key model performance metric is similar to what we achieved previously.

To sum it up, if you're doing any sort of modelling activity on a large dataset containing lots of variables, it is a good practice to perform PCA on that dataset first, reduce the dimensionality and then go ahead and create the model that you wanted to make in the first place. You are advised to perform PCA on the datasets that you worked on in Linear Regression and Clustering as well, to see how it makes our job easier.

In some cases, you can select a particular amount of variance that you want to be explained by the Principal Components of the transformed dataset. PCA automatically chooses the appropriate number of components on its own and proceeds with the transformation.

Those were some important points to remember while using PCA. To summarise:

- Most software packages use SVD to compute the principal components and assume that the data is **scaled and centred,** so it is important to do standardisation/normalisation.
- PCA is a **linear transformation method** and works well in tandem with linear models such as linear regression, logistic regression etc., though it can be used for computational efficiency with non-linear models as well.
- It should **not be used forcefully to reduce dimensionality** (when the features are not correlated).

You learnt some important shortcomings of PCA:

- PCA is limited to linearity, though we can use **non-linear techniques such as t-SNE** as well (you can read more about t-SNE in the optional reading material below).
- PCA needs the components to be perpendicular, though in some cases, that may not be the best solution. The alternative technique is to use **Independent Components Analysis. ICA goes in any direction that has max variance. Use it when data set is very small as it's very slow.**
- PCA assumes that columns with low variance are not useful, which might not be true in prediction setups (especially classification problem with a high class imbalance).