# Principal Component Analysis for All

# Part - I

Principal Component Analysis technique is often considered as an unsupervised learning method to extract information from given data. It is also used as a tool to increase model level signal-to-noise ratio in supervised learning method. In this article, it is discussed as a tool to improve the signal-to-noise ratio.

Principal component analysis is one of the standard techniques of feature extraction and data compression. This technique is often introduced in data science world as a method of dimensionality reduction (data compression). Though that is one of the most useful purpose of this technique, it's main objective is to improve signal-to-noise ratio (SNR), secondary objective is dimensionality reduction. It is effective when distributions between independent attributes show strong linear relations (collinearity).

Often this topic is introduced to new comers with a good amount of matrix algebra, vector algebra and terms like Eigen vectors, Eigen values etc. Though technically perfectly valid approach, many newcomers who do not have prior exposure to the required mathematical constructs are often left high and dry on various aspects of this topic. This article is an attempt to explain these topics at a conceptual level with very minimal use of the underlying mathematical approach.

Some of the concepts that we need to understand before we get into the PCA technique include –

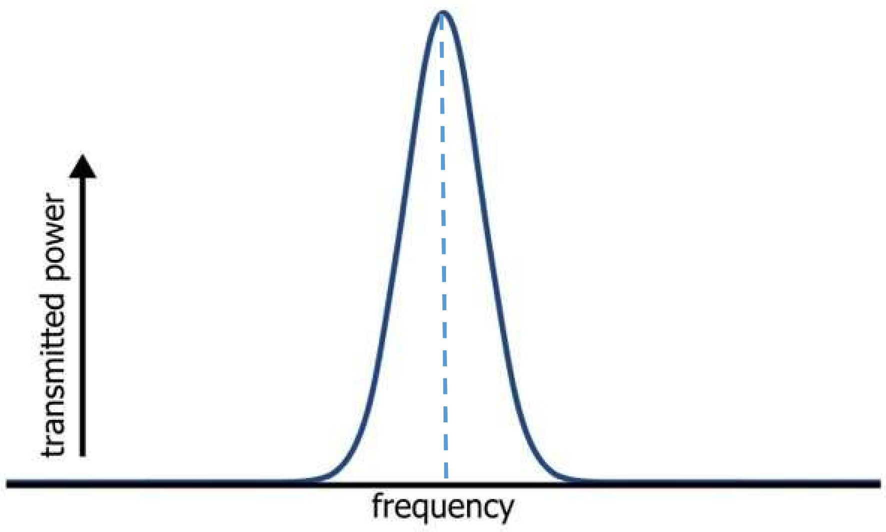
1. Information and spread
2. Covariance a measure of information in multi-variate feature space
3. Directions in features space with maximum information
4. Feature space rotation to get principal components / Eigen vectors and Eigen values
5. Projecting data from original feature space into the rotate space
6. Signal To Noise Ratio (SNR)
7. Impact of PCA on SNR
8. Dimensionality Reduction

Conceptual understanding will help evaluating the usefulness of the technique in a given situation by understanding the advantages gained and the cost of the disadvantages.

**<u>Information is in the spread</u>**

One often comes across this phrase when studying data science topics. To explain this let us look at a day-to-day example (from the 70's ☺)…. Those days when FM (Frequency Modulation) was not yet the standard, we used to listen into the AM(Amplitude Modulation) band of the radios sets. One of the most popular station was the Vividhbharati. Assume it was transmitted at 104.0 AM (do not recall what the frequency used to be). One would notice that the radio sets captured the loudest and clearest signal from this station when the sets were tuned at 104.0 AM. However, one would still get the signals at slightly above this frequency and below this frequency. As you move the tuning knob further and further away on either side, the quality of the signal deteriorates and gets jumbles with unwanted signals (noise).

That means the transmitted signal which is supposed to be at an expected frequency of 104.0 AM is available at the radio set at 104.0 ± i.e. in a spread. With the signal strength / clarity inversely proportional to the spread. Why do the signals spread out? A technical topic beyond the scope but could be due to the various factors between transmission and reception such as bouncing of buildings (reflections), refractions, atmospheric conditions etc.

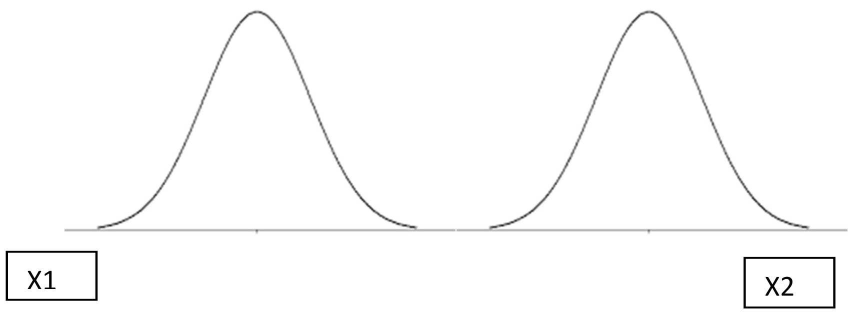


Thus the signals of our interest were available in a spread. If the radio sets were capable of capturing signals only when perfectly tuned to 104.0 AM, then we would listen to the station in breaks! Instead, all the radios consist of a bandpass filter that allow frequencies in certain range to pass thru and attenuate others. In the process they minimize the noise while maximising the signal. The signal is further amplified using amplifiers and we enjoy an uninterrupted program.

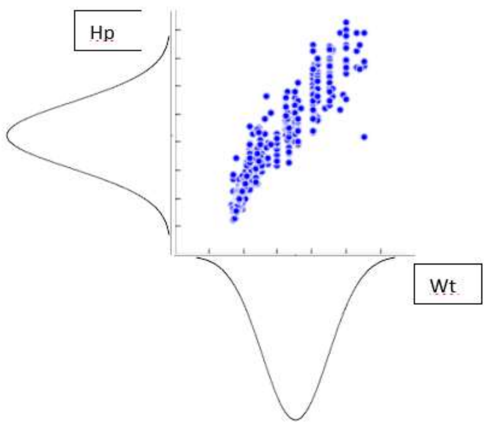
Thus in electronics, we say the information (station signals of our interest) is in the spread. Coming to data science, we know spread in one dimension is measured in terms of variance. Let us take this discussion into a two dimensional feature space as shown below.

## <u>Covariance a measure of information in multi-variate feature space</u>

Let one be denoted by X1 and other by X2 with their respective information spread. Let us assume both have similar spread which is not necessary.



X1 and X2 could be any variables such as “horse power” and “weight” of a car in car-mpg data set of UCI.  A scatter plot between the two dimensions looks as shown below

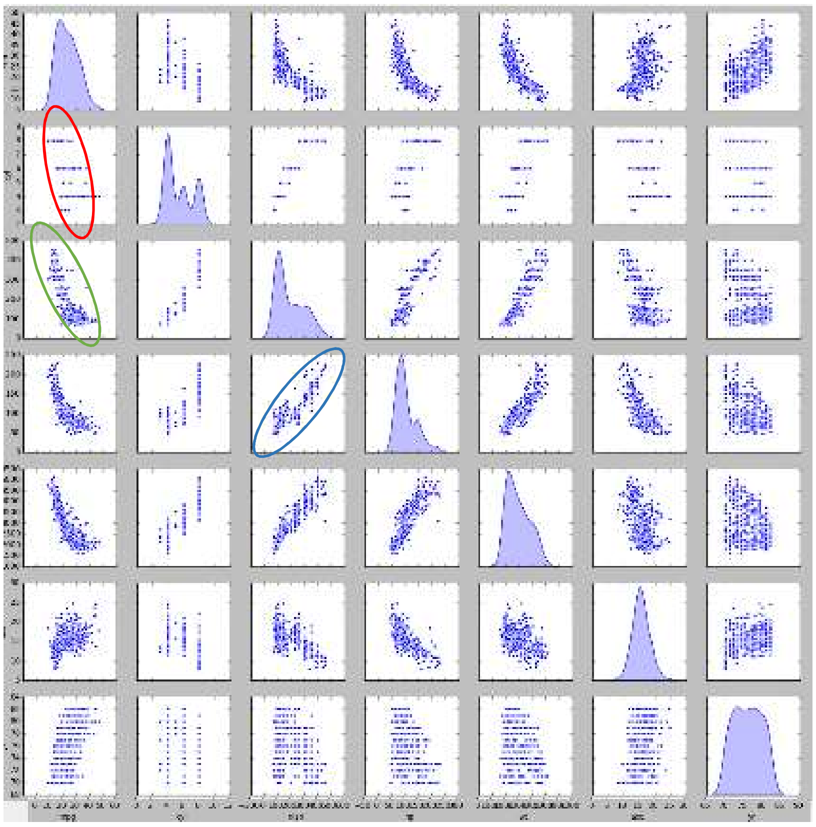


In the scatter plot, the two dimensions of “Hp” and “Wt” contain information in their respective spread / variance. But there is one more rich maybe richer information available in form of the covariance in the feature space! If individual variance contain useful information, would not covariance also contain information? The information that covariance contains is how these two dimensions interact or influence one another.

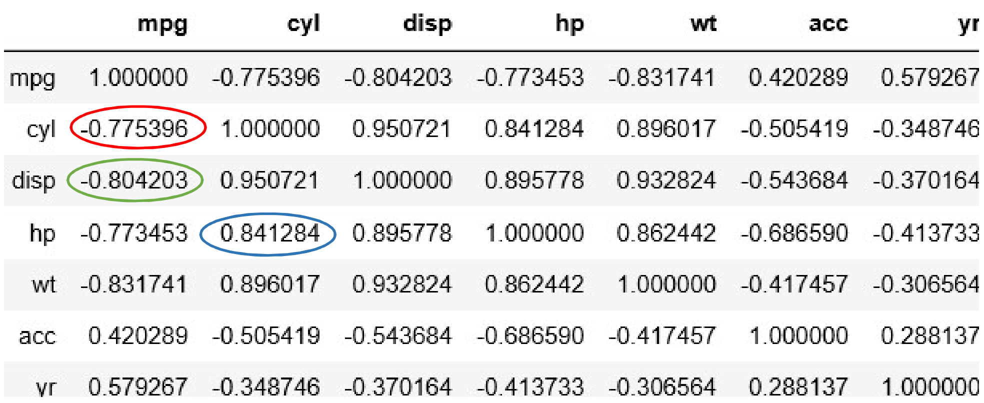
When we build models and provide the various dimensions as inputs to determine the target variable we miss out on the information contained in the feature space!  We do not feed the information in the covariance to the algorithm building that we use to build the model!

Do we not use the expression – model.fit(X_train, y_train) where X_train is only the individual dimensions such as the “Hp” and the “Wt” ? In other words, model.fit(Hp, Wt,  y_train).

When the feature space shows strong covariance between dimensions (excluding the target variable), it is an indication of presence of information in the feature space over and above the useful information in the individual dimensions as spread. For e.g., in the pairplot of the car-mpg data set and the corresponding correlation matrix, it is clear that we have lot of information available in the multi variate feature space.



The same information obtained thru the df.corr() function shows us the magnitude of the information in form of correlation values that is available in the multi variate feature space.



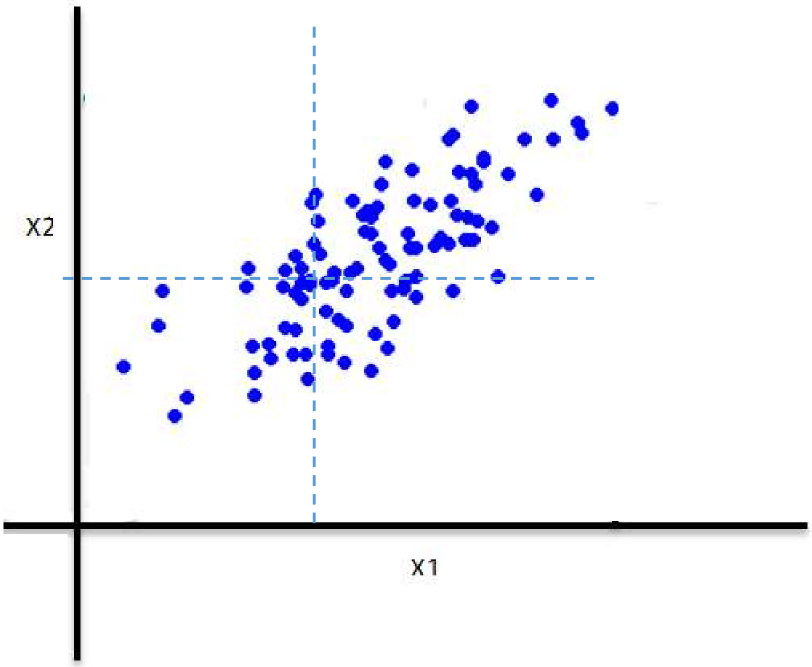
| | mpg | cyl | disp | hp | wt | acc | yr |
|---|---|---|---|---|---|---|---|
| mpg | 1.000000 | -0.775396 | -0.804203 | -0.773453 | -0.831741 | 0.420289 | 0.579267 |
| cyl | -0.775396 | 1.000000 | 0.950721 | 0.841284 | 0.896017 | -0.505419 | -0.348746 |
| disp | -0.804203 | 0.950721 | 1.000000 | 0.895778 | 0.932824 | -0.543684 | -0.370164 |
| hp | -0.773453 | 0.841284 | 0.895778 | 1.000000 | 0.862442 | -0.686590 | -0.413733 |
| wt | -0.831741 | 0.896017 | 0.932824 | 0.862442 | 1.000000 | -0.417457 | -0.306564 |
| acc | 0.420289 | -0.505419 | -0.543684 | -0.686590 | -0.417457 | 1.000000 | 0.288137 |
| yr | 0.579267 | -0.348746 | -0.370164 | -0.413733 | -0.306564 | 0.288137 | 1.000000 |

**Note:** Correlation is covariance expressed in terms of standard deviations. Hence, both give the same information about the feature space. Pairplot gives it visually while correlation matrix gives in quantitative terms. One can also use np.cov() to get the same information in form of covariance matrix

Thus when we do model.fit(X_train, y_train) where X_train refers to all the original columns of the dataframe, we miss out lot of information in the feature space while building a model. How do we make this covariance information available to the algorithm?

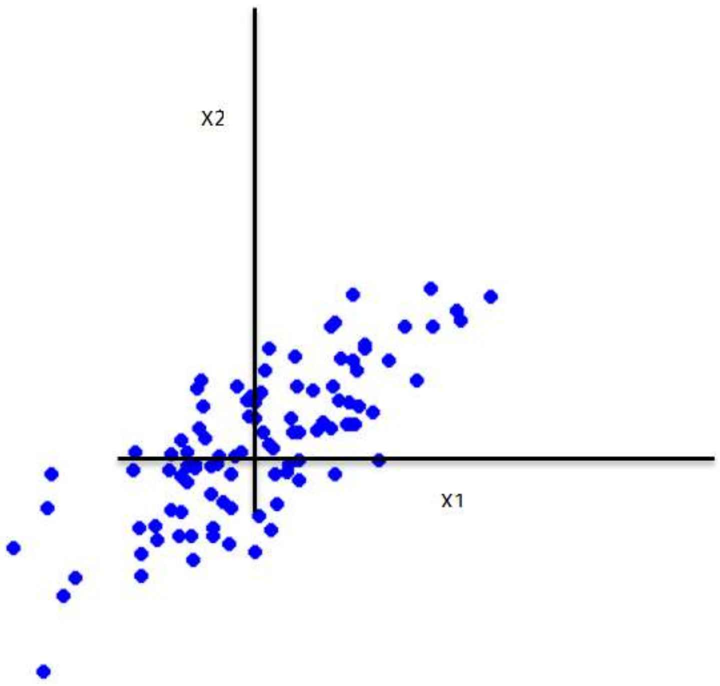
## Directions in features space with maximum information

Let the two original columns have a correlation as shown below.

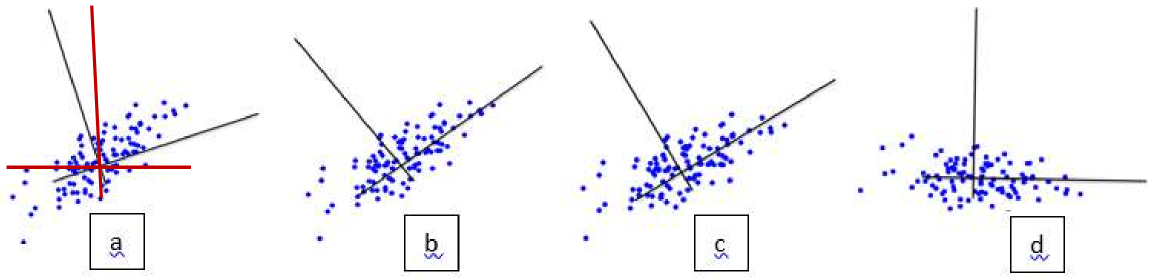


To exploit the covariance information, we first centre the data on the x1bar and x2 bar (shown by the blue dashed lines respectively). For this we use the Zscoring function.

Zscoring is also known as centring of data because the effect of Zscoring is to shift the origin of the coordinates to the point where the x1bar and x2bar lines meet.



Next we rotate the two coordinates i.e. the feature space using a mathematical transformation. This can be done using the Numpy linear algebra function called eig() which takes as input the covariance matrix. The intermediate result of this operation is –



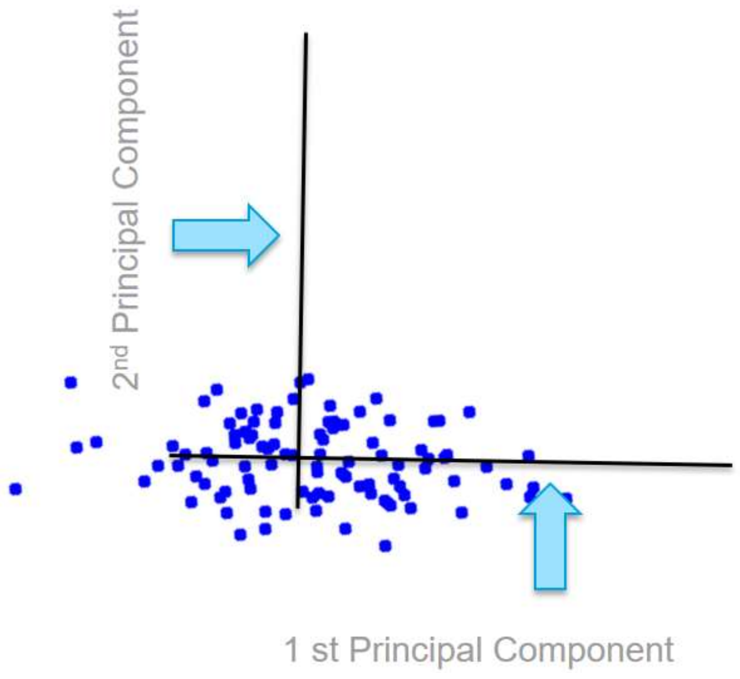
In the above sequence from left to right, in the original feature space of X1, X2 (shown in red), the directions with max information is searched and when found (shown in diagram c), the X1,X2 space is replaced with principal component feature space (PC1,PC2 in black) as shown in diagram d. We say the feature space is rotated to PC dimensions. The first PC dimension captures maximum information followed by next which is perpendicular to first (orthogonal always)

## <u>Feature space rotation to get principal components / Eigen vectors and Eigen values</u>

Once the direction of maximum spread in the original X1, X2 feature space is found, the two new directions in the image 'c' are considered as the new dimensions and they are called the principal components. The one in the direction of maximum spread is principal component 1 (PC1) or Eigen vector 1 (EV1) and the other perpendicular line is principal component 2 (PC2) or Eigen vector 2 (EV2).

The new feature space (after rotation) thus consists of PC1/EV1 and PC2/EV2 as dimensions (shown in diagram below). <u>From now on, we will call these new dimensions the PC dimensions.</u>

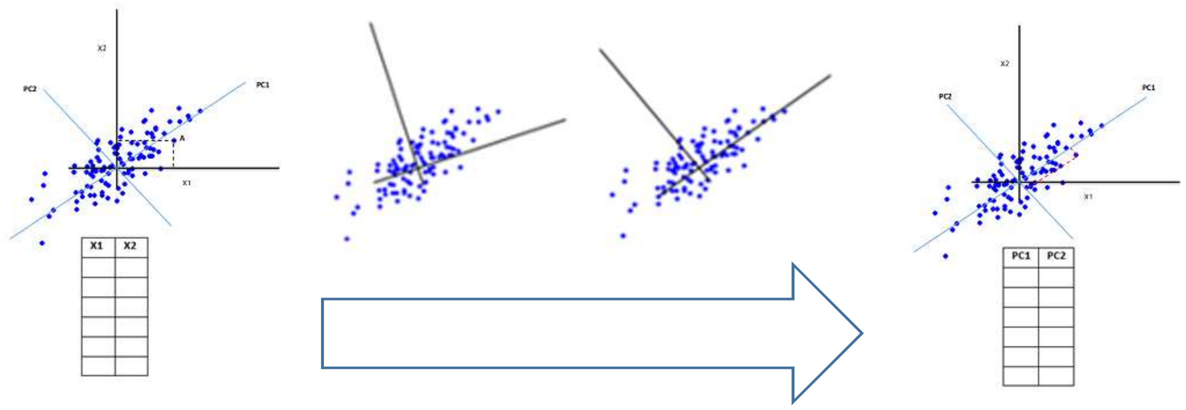
Observe that in this new feature space of PC1 and PC2 there is no covariance (ignoring the correlation due to noise) between the two dimensions. That means all the information is absorbed into the two dimensions. No information is left unused if we build models using PC1 and PC2 in this case.



The amount of information absorbed by the PC dimensions is represented by the spread of the data across these two dimensions. The spread is also known as the Eigen Value.

## Projecting data from original feature space into the rotate space

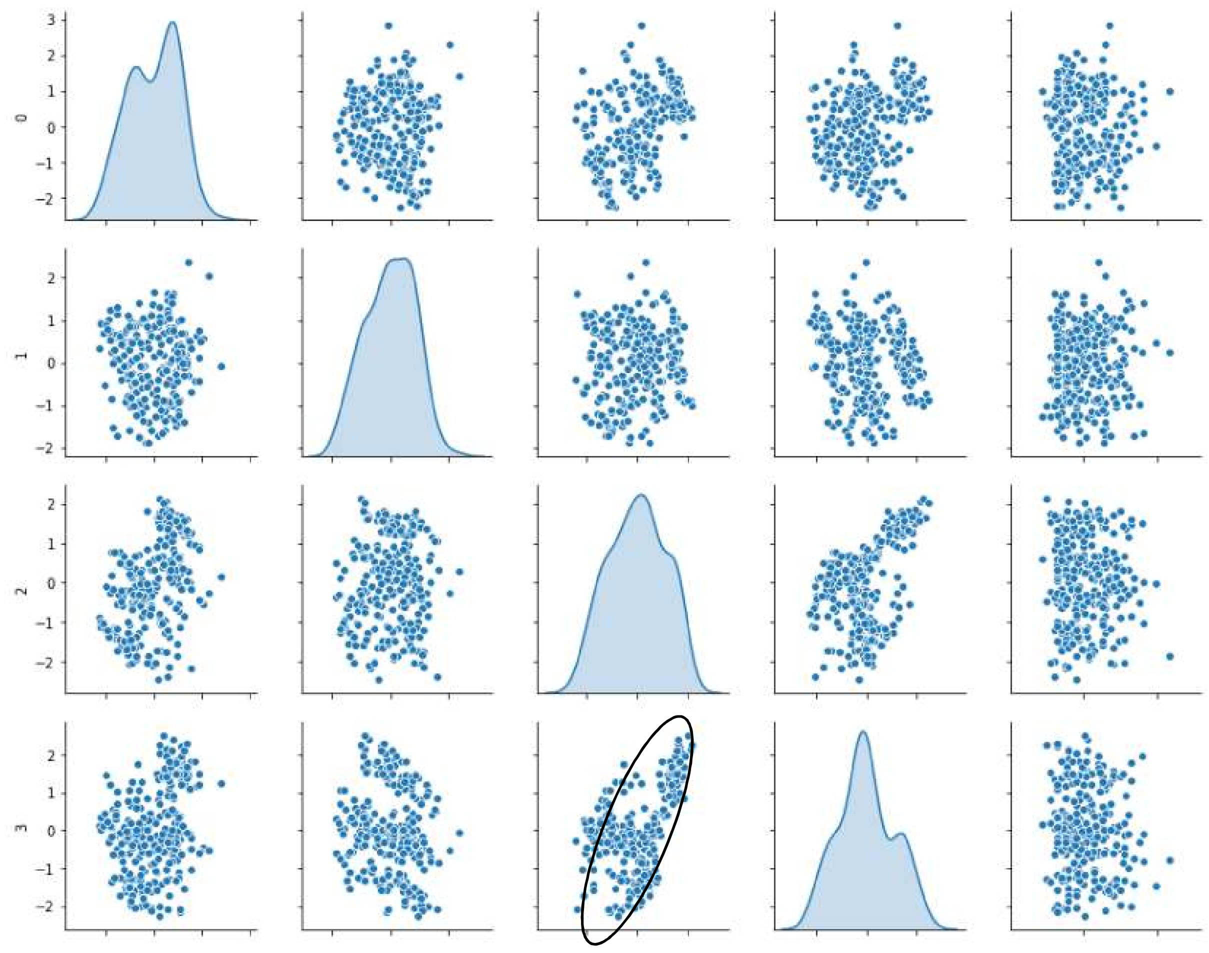
Once the PC1 and PC2 are obtained, the data (which is still available in our data frames in terms of X1, X2 coordinates) have to be transformed into PC1 and PC2 terms and represented by a new data frame.



In the diagram above, the data is provided in terms of two original dimensions X1 and X2 and represented as the dataframe on the left. On application of the Eigen transformation (represented by the arrow), the two directions of maximal information in original X1,X2 space is found and represented as PC1, PC2. But the data is still represented by original dataframe!

The next step will be to represent the original data now in terms of PC1 and PC2 and thus create the new dataframe shown on the right. Where all the data points are expressed in terms of PC1, PC2 dimensions. One data point (let this be first record in the data set) is shown undergoing this transformation. All the data points have to undergo this transformation for which we use a dot product operation between the original data set and the PC dimensions

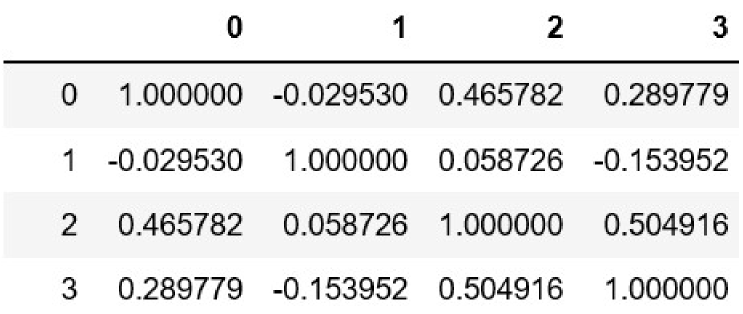
A pair plot between PC dimensions will not show any collinearity (except for that caused by noise) and the correlation values will be close to zero as shown below.



Note: Only four PCs are shown in the picture above.

<u>Observations from the pairplot of the PC dimensions</u> -

1. That the correlation visible as scatter plot between the PC dimensions look almost <u>like a cloud is an indication of the fact that between the PC dimensions there is no collinearity i.e. no covariance and hence no information in the feature space</u>.
2. Some weak collinearity is observed between the PC3 and PC4 (circled). This is likely due to noise i.e. a statistical chance relationship. Not true relation
3. The same information can also be represented as a covariance or correlation matrix…



| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1.000000 | -0.029530 | 0.465782 | 0.289779 |
| 1 | -0.029530 | 1.000000 | 0.058726 | -0.153952 |
| 2 | 0.465782 | 0.058726 | 1.000000 | 0.504916 |
| 3 | 0.289779 | -0.153952 | 0.504916 | 1.000000 |

4. The correlation matrix between the PCs show a very small correlations which are likely to be by chance.
5. Correlation value can range from -1 to + 1. Closer the value to 0, more likely the correlation is insignificant

Now when we build models using model.fit(projected_x_train, y_train) where projected_X_train is the data from the second dataframe i.e. data in PC dimensions, we are supplying all the information (in form of eigen values) to the algorithm to build the model on.

We have no information left unused in the new feature space made of the PC dimensions. This can be confirmed by looking at the correlation matrix between the PC dimensions. All the corrrelations are close to 0.

## Signal To Noise Ratio(SNR)

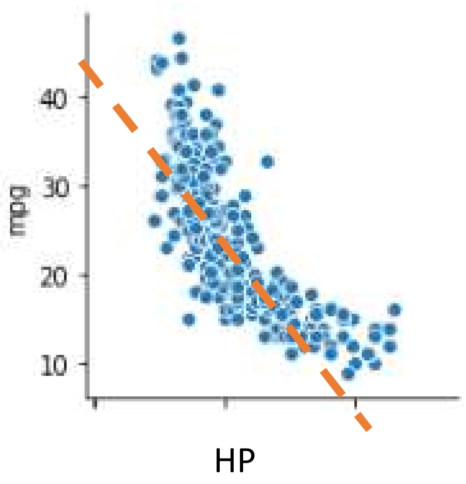
To understand this ratio we first need to understand what signal is and what noise is. Signal is also known as information. In the context of the example this article started with, that is the ratio station of choice, the signals of that station is information while the background signals that are not related to this station, is noise in that context. Finest of the fine radio sets with most accurate tuning will still generate noise in the output. But the signal content (magnitude) will be so large compared to the magnitude of the noise that we will be completely ignorant of the noise.

In the context of data science, SNR is the characteristic of a model. Models that have a good SNR will be highly accurate and generalize. Models that have poor SNR will have low accuracy when tested against unseen data. They may perform very well against training data though, indicating the case of over fitment.

What leads to the SNR in a model? There are two main factors that impact the SNR -

1. The columns (a.k.a attributes) that go into building of the model
2. The structure / complexity of the model

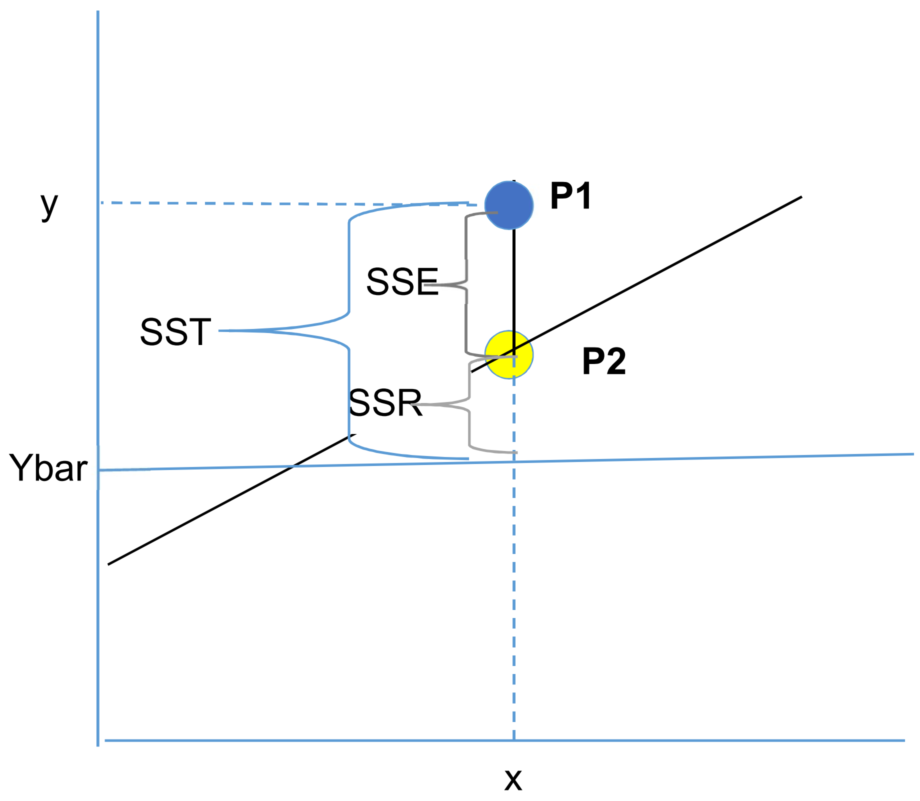
Let us consider the car-mpg dataset again. This time we will take the mpg(target) and Hp(predictor) variables for this discussion. All the conceptual discussion will be the same for any other independent variable in this dataset or any other dataset.



Imagine the best-fit-line be the orange dashed line (linear regression modelling). It shows the relationship between horse power (HP) and miles per gallon (mpg) as negative but highly correlated.

How good this linear model is, is decided by a metric called the coefficient of determinant (R^2). Let us imagine this to be 86%. That means this model is able to explain 86% of the total covariance between the 'mpg' and the 'hp'. Rest of the covariance i.e. 14% is unexplainable. This unexplainable variance is called the noise.

There is another metric called the SSE (Sum of Squared Errors) that we use in linear regression models. This metric is used to find a model from among the infinite possibilities that will give us the max R^2 (coefficient of determinant).



In this example, y is the target variable and x is the predictor. Simple linear regression reflecting positive correlation between y and x is shown here for ease of discussion.

1. For a given value of X, the predicted value is P2, the yellow one.
2. The actual value of y for the given X in the dataset is P1, the blue one.

The difference between actual (P1) and the predicted (P2) for the given X is the unexplained variance also expressed as Squared Errors. When you consider the unexplained variance across all points in the data set and sum it up it becomes SSE (Sum of Squared Errors)

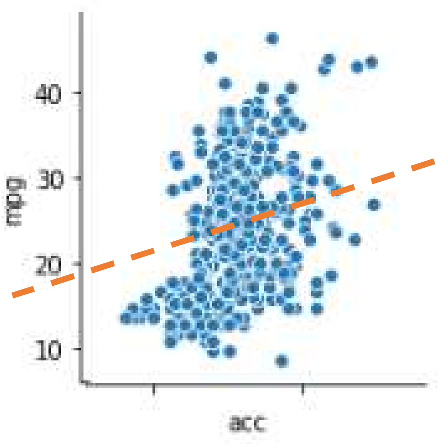
The ratio of SSR/SST is called the coefficient of determinant (R^2). Closer this ratio is to 1 the better the model is because the unexplained variance (SSE) then becomes smaller and smaller. In other words, as the noise decreases, the R^2 increases.

How do we define noise in classification cases? For same values of given variable we have data points belonging to multiple classes and the density of the different classes is similar then we have very noisy data and this is reflected as the Entropy or Gini in Decision Tree algorithms.

In other words, noise is unexplainable variations in the patterns between independent and dependent variables. Theory says that the noise is likely due to other attributes important for the predictions but not considered in the model

**Good Vs Poor Attributes**

All attributes contribute both information/signal and noise to the model. Good attributes contribute more information than noise to the model while poor attributes contribute more noise than information to the model. For e.g. in the car-mpg dataset, the acceleration column (shown below) contributes more noise to the model than information. Hence, it is a poor attribute to predict mpg.

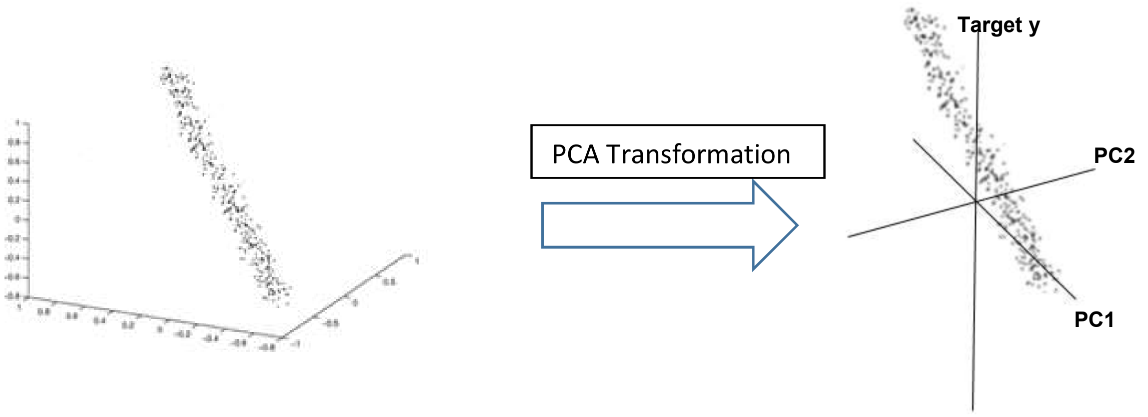


The linear regression between these two come out to be weak and positive (ref the covariance matrix) with magnitude of 0.42. And the scatter plot shows there will be lot of unexplained variance in predictions of mpg based on acceleration. This attribute will contribute more noise to the model than information compared to other attributes such as horse power

## Impact of PCA on SNR

When we apply PCA on a data set to improve SNR, separate the target variable from the predictor variables. PCA is applied on the input predictor variables space to soak up the information available in the feature space in form of covariance between the variables *.

In the following situation, the vertical dimension is the target variable and two horizontal dimensions are the predictor variables. For PCA, project the data onto the floor (separate out the target variable).

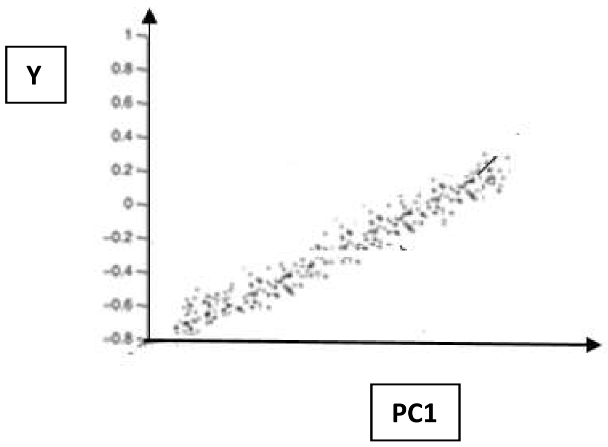


In simpler words, the PCA technique rotates the input space to get the PC1 and PC2. The target columns is not involved in PCA (Remember, we are discussing PCA in the context of supervised methods).

The PC dimensions absorb the covariance in the original feature space. That means the PC dimensions put together have more information content than the original dimensions put together (they ignored the covariance information). Hence, when we build models using PC dimensions, the models are likely to be built on richer information set and hence likely to perform better than model built on original dataset.

## Dimensionality Reduction

When we observe the PC dimensions such as in the figure above, one can notice that the PC dimensions have different degree of spread of data i.e. contain different amount of information. In fact, in the above case, the amount of information in PC2 is very miniscule compared to that in PC1 (the Eigen value of PC2 will be very small in comparison to the Eigen value of PC1). In such a situation we can leave out the PC dimensions with information content below a threshold. Project the original data into the reduced PC dimensions space and build models. In the diagram above, we can build model between Y and PC1 alone. Drop PC2.



The logic behind dropping PC dimensions is the same signal to noise ratio. PC dimensions would have absorbed both information and noise from the original dimensions. Due to the covariance information content, PC dimensions would have more information than original dimensions. However, the first PC dimension would have absorbed most of the covariance followed by the next PC dimension. As in the above example, PC1 is likely to result in more signal than noise at the model while PC2 is likely to result in more noise than signal given the low information content in PC2. Thus dropping PC2 may give a better model than including it.

**Wrapping up with vital information on PCA**

1. Apply PCA techniques only when collinearity between the original predictor columns is significant. As modelling on original dimensions in that case will leave much of the information unused.
2. Applying PCA in situations where collinearity is small as in case of mpg Vs acceleration, will do no good. The PC dimensions will have same ratio of signal to noise as original dimensions (Why?)
3. PCA technique will return as many PC dimensions as the number of original dimensions. If we had thirty independent columns in original data frame and performed PCA on it, we will get thirty PC dimensions and 30 Eigen values reflecting the amount of spread captured on each of the 30 PC dimensions respectively

4. Each PC dimension is a composite of all the original dimensions. That is every original dimensions plays role in defining the PC dimensions. Hence, giving physical meaning / interpretation of the PC dimension become impossible. For e.g. PCA on horse power and weight will result in PC1 and PC2 where each PC is some composite of horse power and weight. But there may be no physical meaning of such PC dimensions.
5. Models build on PC dimensions are considered as black box. We cannot interpret the models in terms of original meaningful columns
6. Mathematically it is possible to recover all the original dimensions from the PC dimensions as long as no PC dimensions is dropped. If a PC dimensions has been dropped then the original dimensions cannot be recovered
7. Though a popular technique for dimensionality reduction and SNR maximization, in regression case, Lasso is also popular for dimensionality reduction resulting in better SNR at model level. The resulting Lasso model is white box!
8. Simple log transformation of the attributes showing strong correlation with target (mpg) but visually not a line, will also help reduce noise. This is not discussed in this post
9. PCA can be applied on data sets with categorical data. Refer to https://arxiv.org/abs/0711.4452 and https://stackoverflow.com/questions/40795141/pca-for-categorical-features
10. If PCA is done separately on continuous variables alone, then modelling on PCA should be done separately i.e. should not combine PCA dimensions with other original dimensions to model. They are two different feature spaces.

A very good introduction to PCA - http://setosa.io/ev/principal-component-analysis/

**Note:** - When we use PCA from Sklearn's decomposition package, all the steps from generating covariance matrix to the transformation of original data to PC dimensions, all happen under the hood.

-------------------------------------XXXXXXXXXXXXXX---------------------------------

# Principal Component Analysis For Mathematically Inclined Part - 2

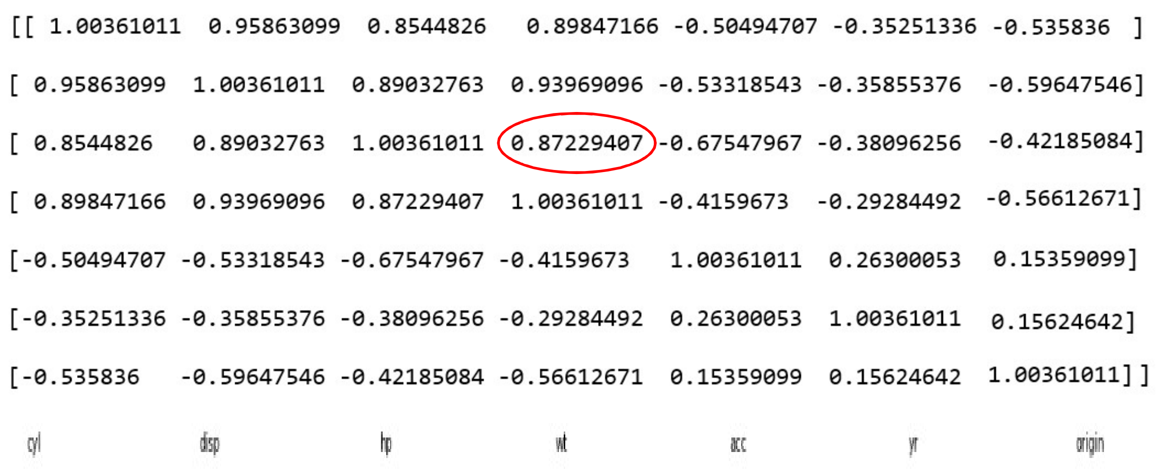
In this post, I plan to introduce the underlying mathematical flavour of PCA without going too deep into it as my own mathematical skills are limited. What we covered conceptually can be understood in terms of matrices and matrix algebra without becoming too mathematical. This understanding is not mandatory for successful usage of the PCA techniques. However, if one is curious, the underlying mathematical constructs is not too difficult to understand.

We apply PCA technique only on those datasets where we notice strong collinearity between the predictor variables. The objective is to increase SNR and if need be, dimensionality reduction.

Let us consider the car-mpg dataset for this discussion. The pairplot shows significantly strong collinearity between some of the predictors. Let us represent the pairplot I quantitative terms such as using the covariance matrix.

$$C = \begin{bmatrix} \operatorname{cov}(X,X) & \operatorname{cov}(X,Y) & \operatorname{cov}(X,Z) \\ \operatorname{cov}(Y,X) & \operatorname{cov}(Y,Y) & \operatorname{cov}(Y,Z) \\ \operatorname{cov}(Z,X) & \operatorname{cov}(Z,Y) & \operatorname{cov}(Z,Z) \end{bmatrix}$$

This can be generated easily using df.cov() function where df is the dataframe of predictor variables. We get the following for the car-mpg.df



```
[[ 1.00361011  0.95863099  0.8544826   0.89847166 -0.50494707 -0.35251336 -0.535836  ]
 [ 0.95863099  1.00361011  0.89032763  0.93969096 -0.53318543 -0.35855376 -0.59647546]
 [ 0.8544826   0.89032763  1.00361011  0.87229407 -0.67547967 -0.38096256 -0.42185084]
 [ 0.89847166  0.93969096  0.87229407  1.00361011 -0.4159673  -0.29284492 -0.56612671]
 [-0.50494707 -0.53318543 -0.67547967 -0.4159673   1.00361011  0.26300053  0.15359099]
 [-0.35251336 -0.35855376 -0.38096256 -0.29284492  0.26300053  1.00361011  0.15624642]
 [-0.535836   -0.59647546 -0.42185084 -0.56612671  0.15359099  0.15624642  1.00361011]]
    cyl          disp         hp           wt           acc          yr           origin
```

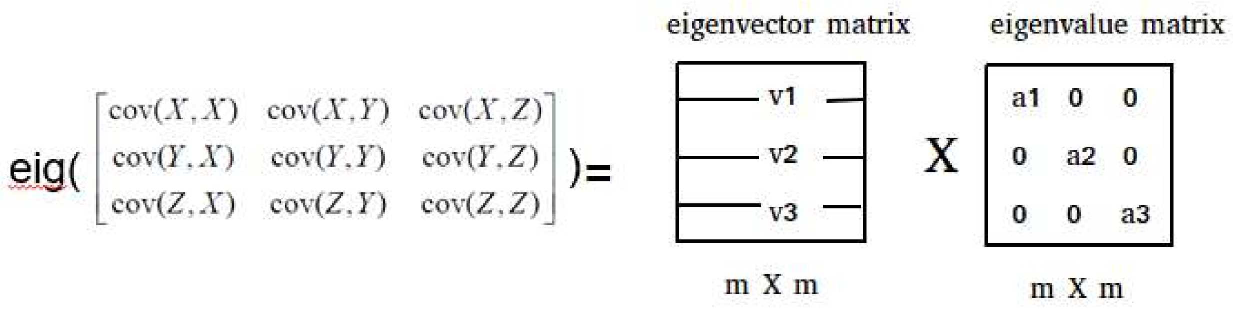
This covariance matrix shows quantitatively the magnitude of covariance between the seven predictor variables. For e.g. between “Hp” and “Wt” shown in red circle. This covariance matrix gives the same information that the pair plot diagram gives visually.

This covariance matrix is supplied as input into a function such as Numerical Python Linear Algebra’s eig function. This function create two matrices out of this. Let us call it EVec and Eval matrices respectively. The beauty is when we multiply Evec and Eval, we get back the original covariance matrix.

### Evec and Eval Factor Matrices

The Eval matrix is special. If you observe this matrix it will have strong values in the diagonal and off diagonal will be 0 or a very small value (caused by noise in the data)

If there were seven columns in the original dataframe, there will be seven rows both in the Eval and Evec matrices. Each row of the Evec representing an Eigen vector or Principal component while the corresponding row in Eval will have diagonal value reflecting the Eigen value of the corresponding Eigen Vector.

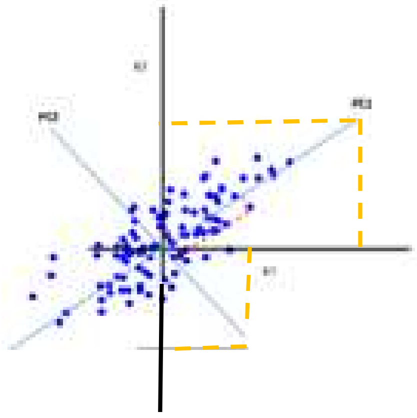

eig(
$\begin{bmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{bmatrix}$
)=
eigenvector matrix
v1
v2
v3
m X m
X
eigenvalue matrix
a1 0 0
0 a2 0
0 0 a3
m X m

In this example m=3

Ref: https://www.sciencedirect.com/topics/mathematics/matrix-decomposition

Eigen vector matrix is the definition of the principal components in terms or original dimensions. For e.g. in the figure below, the Eigen vector 1 or PC1 and Eigen vector 2 or PC2 will be defined in terms of X1 and X2 as shown by the green dashed lines

Thus every PC dimension represent a direction in the original mathematical space and the direction will be specified in terms of values on the original dimension



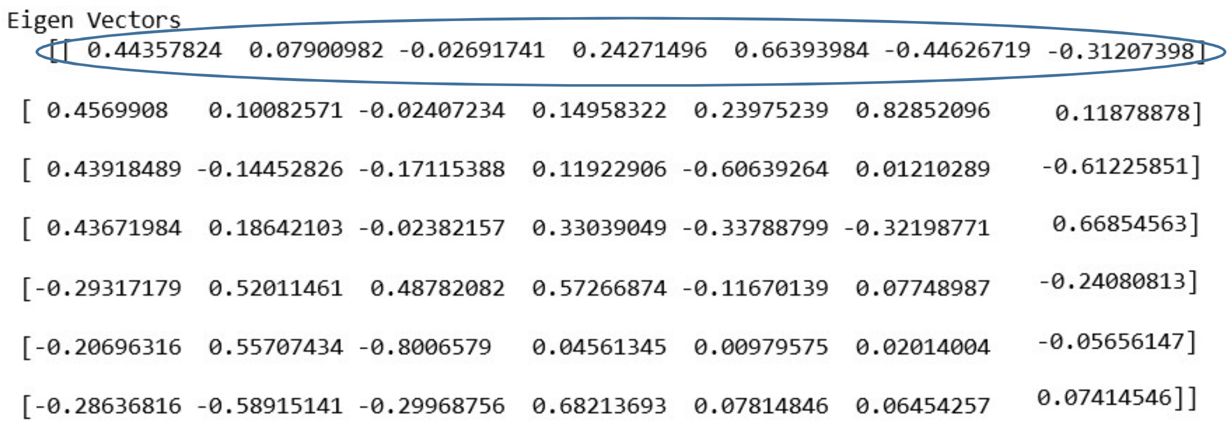
If there are two original dimensions, there will be two principal components defined in terms of the original dimensions. The matrix representing the PC components a.k.a Eigen vector matrix will in this case be a 2X2 matrix

Evec = $\begin{bmatrix} 0.94868 & -0.31623 \\ 0.31623 & 0.94868 \end{bmatrix}$ (the numbers are only for example)

The other matrix will contain the magnitude of spread in the direction of each of the principal components respectively.

Eval = $\begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$ (the numbers are only for example)

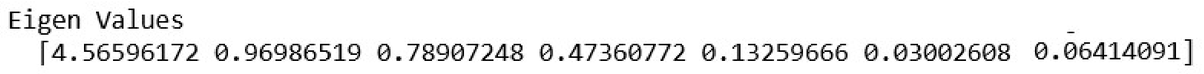
Thus, in the case of car-mpg dataset which has seven columns will result in seven PC dimensions after the transformation. The seven PC dimensions will be expressed in the context of the seven original dimensions. Hence the matrix will look like below –



```
Eigen Vectors
 [[ 0.44357824  0.07900982 -0.02691741  0.24271496  0.66393984 -0.44626719 -0.31207398]
 [ 0.4569908   0.10082571 -0.02407234  0.14958322  0.23975239  0.82852096   0.11878878]
 [ 0.43918489 -0.14452826 -0.17115388  0.11922906 -0.60639264  0.01210289  -0.61225851]
 [ 0.43671984  0.18642103 -0.02382157  0.33039049 -0.33788799 -0.32198771   0.66854563]
 [-0.29317179  0.52011461  0.48782082  0.57266874 -0.11670139  0.07748987  -0.24080813]
 [-0.20696316  0.55707434 -0.8006579   0.04561345  0.00979575  0.02014004  -0.05656147]
 [-0.28636816 -0.58915141 -0.29968756  0.68213693  0.07814846  0.06454257   0.07414546]]
```

PC1/ Eigen Vector 1 is defined in the first row in terms of the seven original dimensions. There are seven Eigen vectors.

Each Eigen vector would have captured some degree of covariance in the original space. The amount of covariance captured is represented by the Eigen value. Each vector will be associated with Eigen value as shown below.



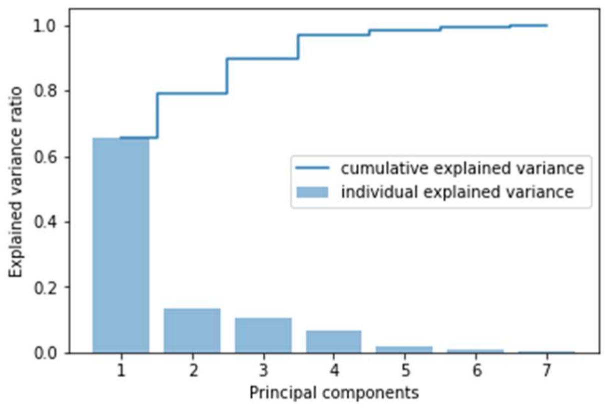
```
Eigen Values
 [4.56596172 0.96986519 0.78907248 0.47360772 0.13259666 0.03002608  0.06414091]
```

**Note:** When Eigen values are printed, the diagonal of the matrix is printed and it looks like a vector rather than a two dimensional matrix.

### PC information content

As you can notice from the Eigen Values vector, not all PCs capture high information. For e.g. the last two Eigen values are too small compared to the first five entries. That means the information content in the last two Eigen vectors is very low.  They may contribute more to noise than information at the model level. We can drop those two and build model with rest five.

To compare the Eigen vectors we use the cumulative information graph.



Some points to notice about this graph –

1. The column chart reflect %age of information captured by each PC dimension
2. The %age of information capture decreases from first PC to last PC dimension
3. The cumulative information content graph (looks like step function) increases by significant amount initially but starts tapering off towards 6th and 7th PC dimension
4. That indicates very minute contribution of the sixth and seventh PC component. Maybe even fifth PC dimension is not relevant
5. Thus all subsequent analysis can be done using only the first four or five PC components

The advantage of dimensionality reduction include –

1. Avoid the risk of curse of dimensionality and thus overfitting
2. Increased SNR as the last few PC dimensions contribute more noise than signal to the model
3. Reduced computational load while model is being built

--------------------------------- XXXXXXXXXXXXXXXXXXXXXX -------------------------------

## Principal Component Analysis

## Code block

## Part - 3

In this section, we will walk through a code to understand the PCA steps discussed above. The section also covers a potential pitfall when transforming data that can lead to confusion when attempting to visually understand the relation between the target and the principal component dimensions. This pitfall does not impact the model but can lead to confusion if not addressed.

Summary (of the pitfall) –

1. When a data is randomly split into training and testing datasets i.e. (X_train, y-train), (X_test, y_test) and the X_train and X_test are transformed using standard scaler or any other tool, the internal index of the records in both the new datasets (X_train_std, X_test_std) change. Though the records physically remain in the same order and their one-to-one correspondence with the y_train and y_test remain intact, the indexes don't match!

2. When the join is done between the X_train_std data and y_train for e.g. Proj_train_data_df and y_train in the code, the join is done using index leading to incorrect joins!

3. In some records it will introduce Nan in the y_train as the y_train does not have the index value which Proj_train_data_df has!. This incorrect join leads to incorrect pairplot between the target and independent variables

For further details, refer to the code….

PCA_car_mpg.ipynb

--------------------------------------XXXXXXXXXXXXXXXXXXXX-----------------------------