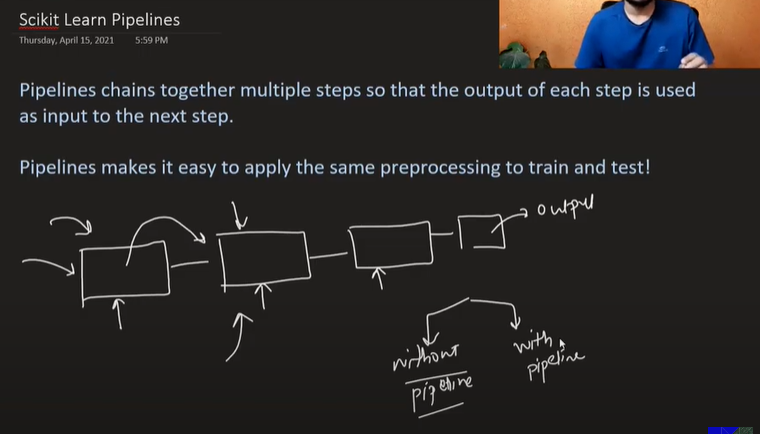
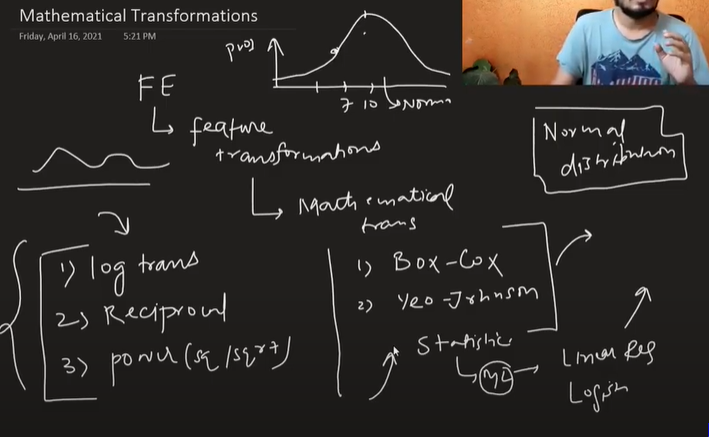
**Machine Learning Pipelines A-Z | Day 29 | 100 Days of Machine Learning**

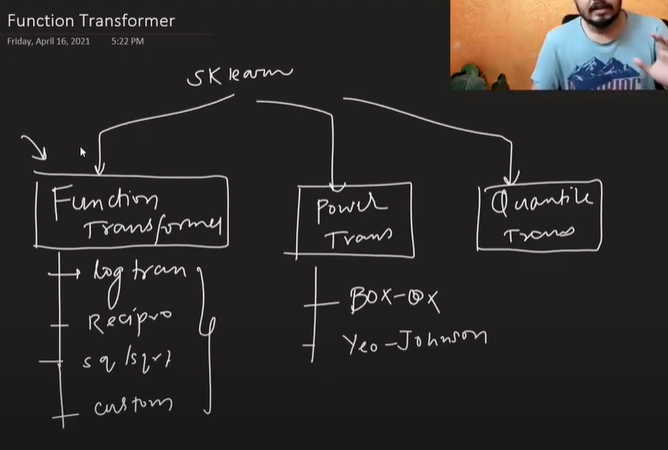


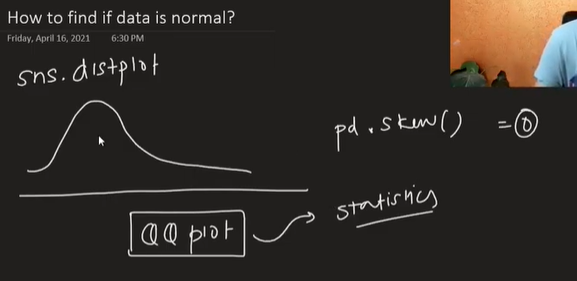
In Scikit Learn Pipelines we can holistically build a pipeline of set of steps that we regularly perform individually such as imputing null values, encoding and training the model

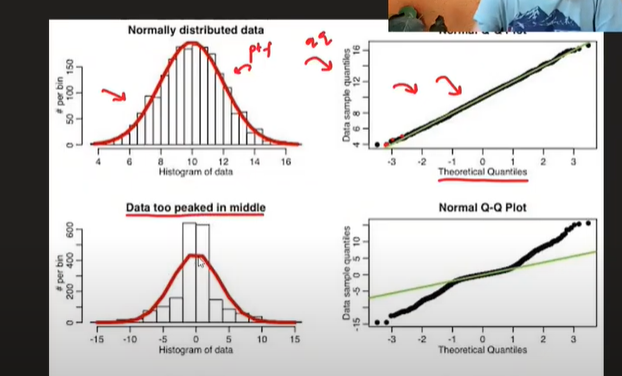
# Function Transformer | Log Transform | Reciprocal Transform | Square Root Transform

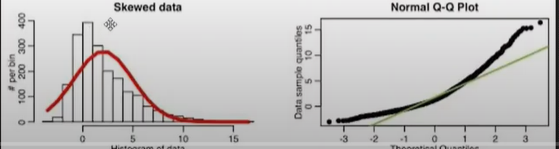


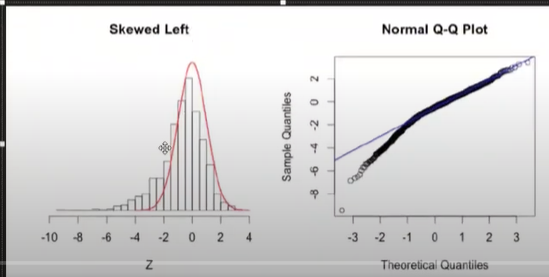
These Feature transformations are used to transform data to normal distribution. Statistical models such as Linear Regression and Logistic Regression demands the normal distribution and hence these feature transformation converts any kind of distribution into normal distribution

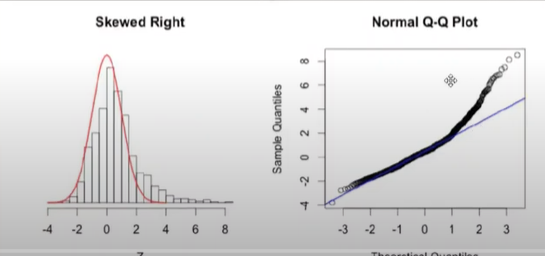


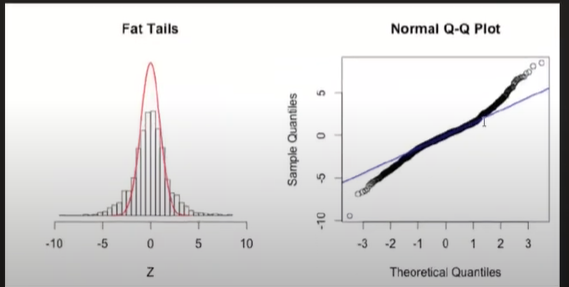


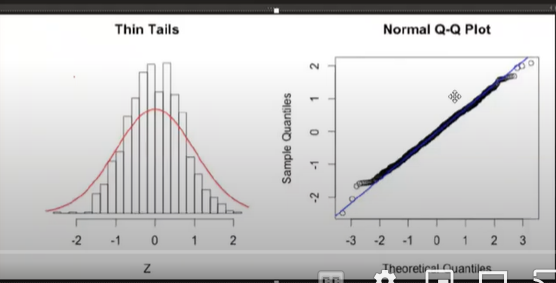








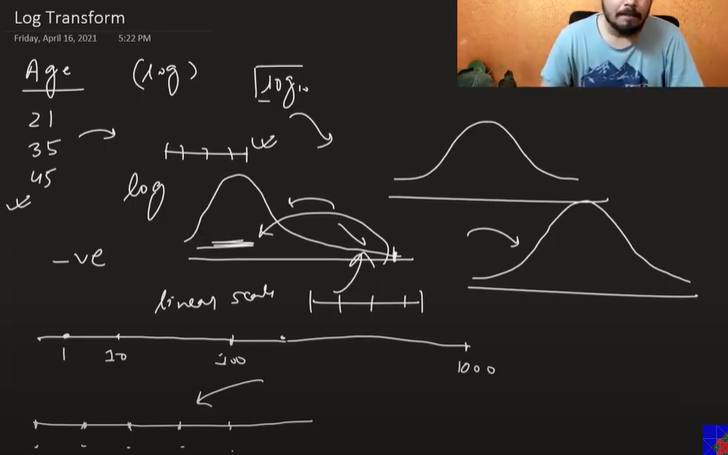


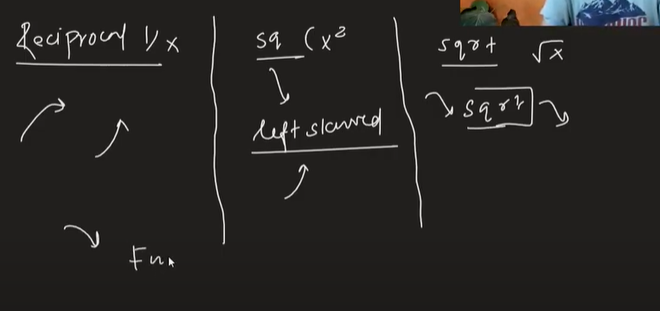


**Log Transformer**

It converts the bigger scales into smaller linear scales in equidistance

It cannot be applied on the negative values but it is useful in the right skewed data





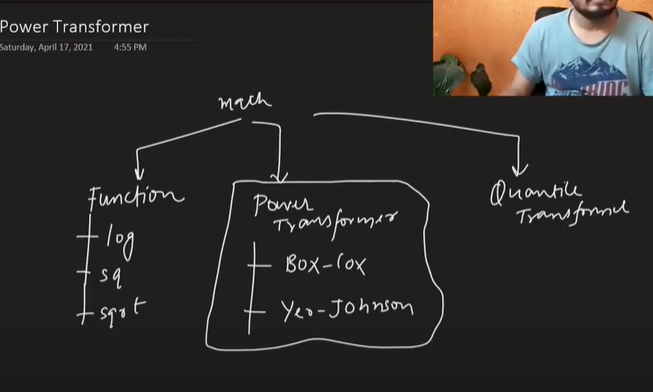
Square transform is used in case of left skewed data

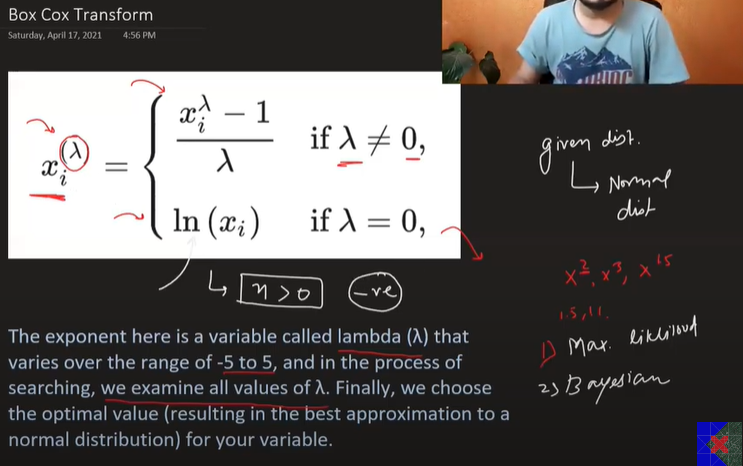
Difference b/w np.log and np.log1p is that 1p adds 1 and then applies log, while the dataset has some zeros we use 1p else we use np.log

Applying log transformation to normally distributed data leads to less accuracy while it doesn’t change results that much if we apply it to tree based models

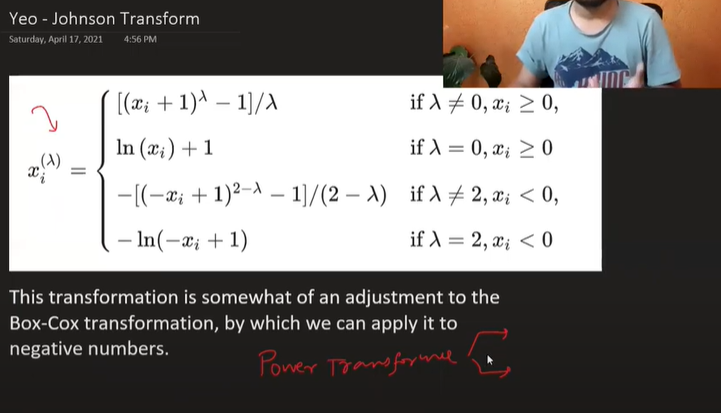
It gives better results in Linear/ Logistic Regression models

# Power Transformer | Box - Cox Transform | Yeo - Johnson Transform





Box cox transformation doesn’t work on zero and negative values

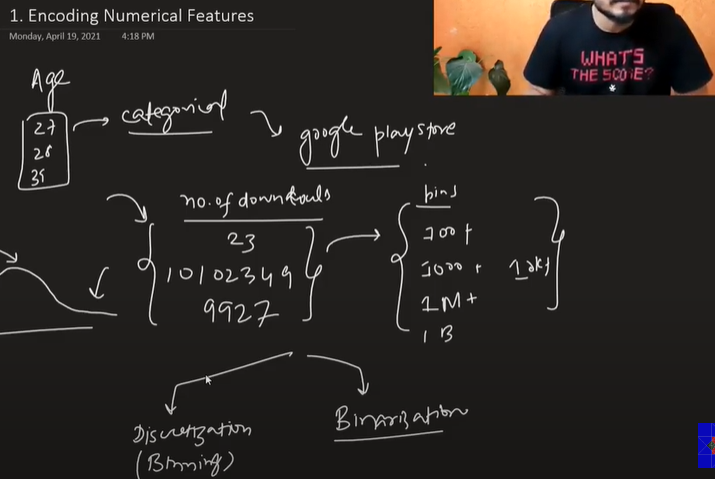


Lambda value is calculated using maximum likelihood or Bayesian stats

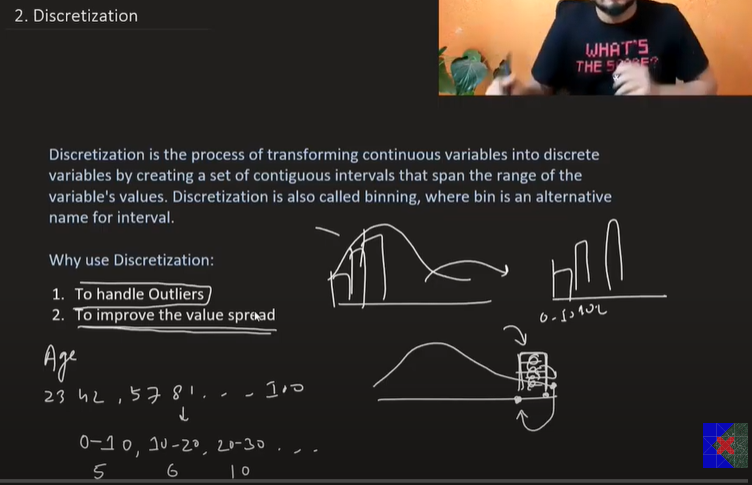
We don’t need to apply standardization separately while applying PowerTransformer()

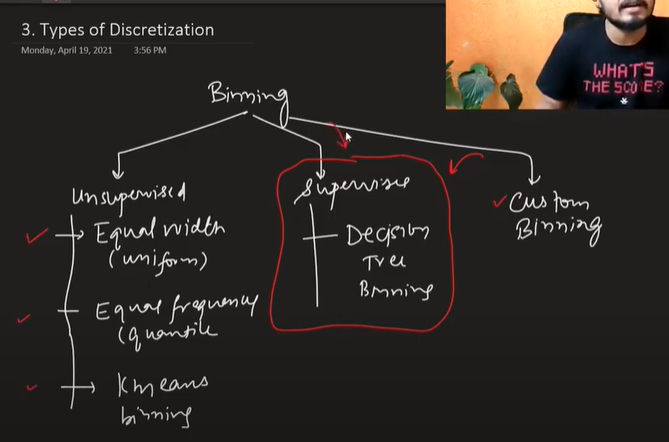
When we are using box-cox transformation, if the values are zero it won’t work. So we add 0.00001 to avoid these scenarios

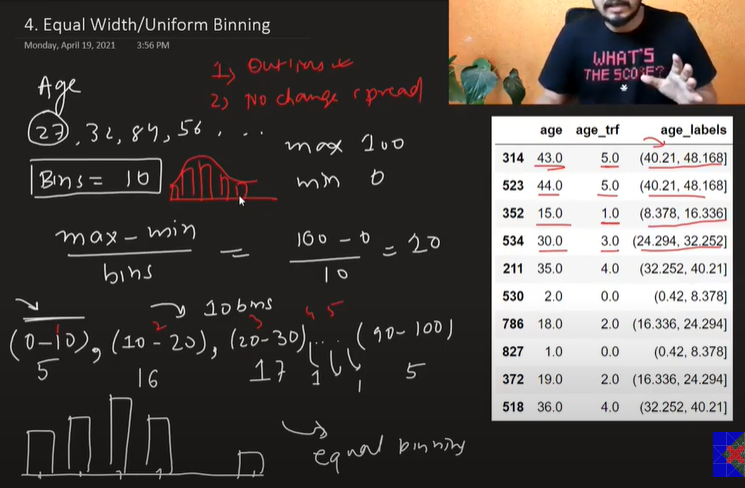
# Binning and Binarization | Discretization | Quantile Binning | KMeans Binning



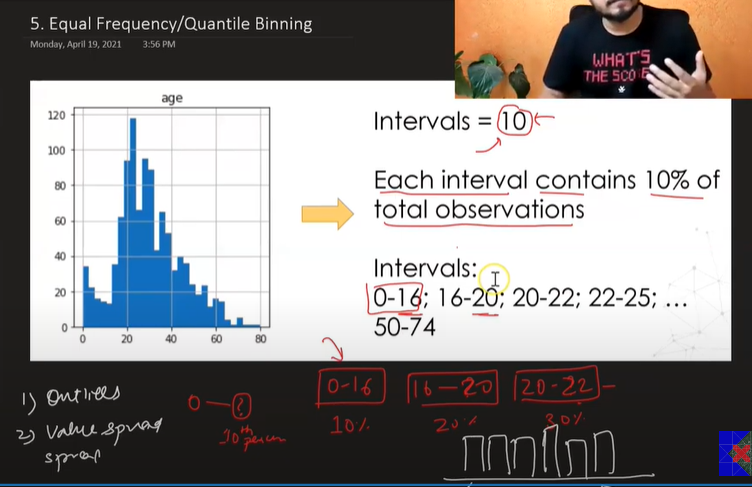
Now in some scenarios encoding the numerical values can do the trick





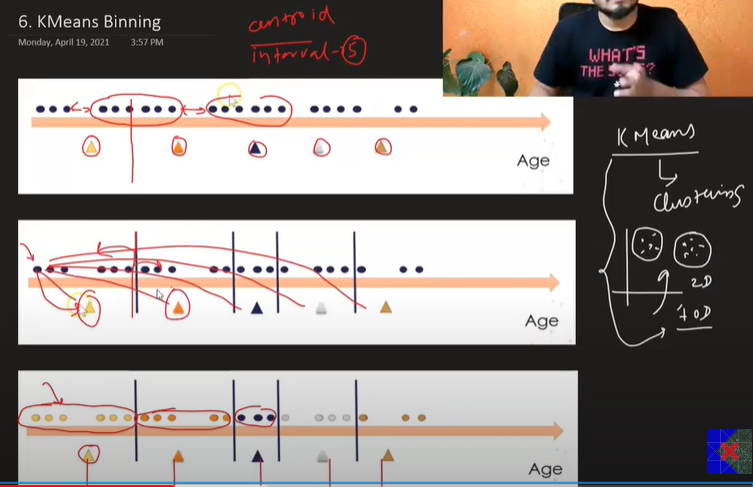


Equal width/ Uniform binning helps to handle outliers and there is no change in the spread



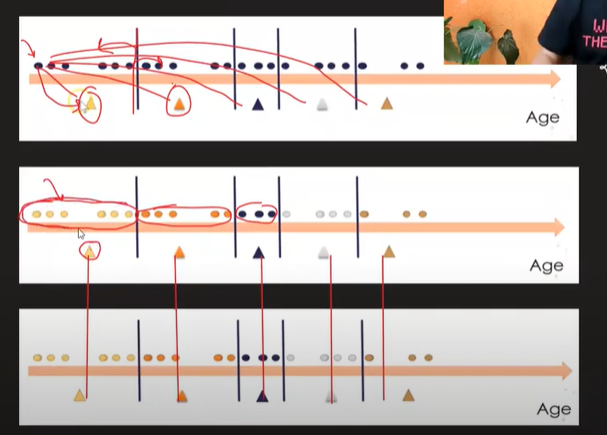
In this type of binning, the bin size may not be equal but each bins stores 10 percentile data. So 10 intervals will score 100 percentile data

This is also robust to the outliers and the makes the spread uniform



This type of binning is used in scenarios where the data is in clusters

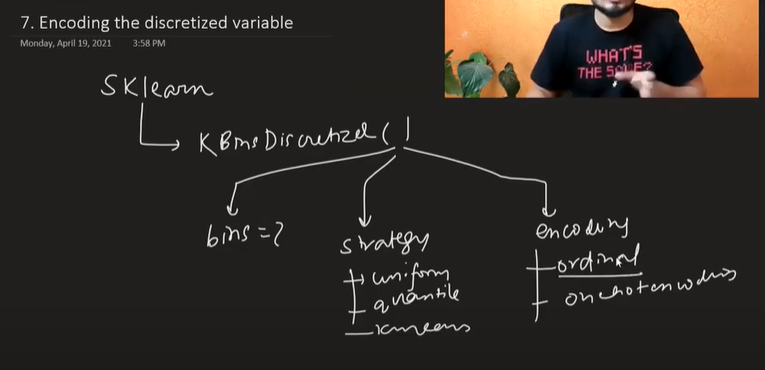
We define centroid and calculated distance of all the points from each centroid. The point which is closer to the respective centroid is considered and that point is mapped in same cluster

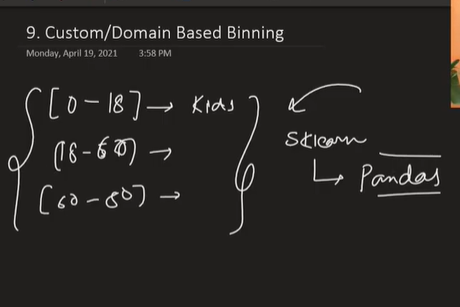


Now when all the clusters are formed, we take the mean of all the points in the clusters. After calculating the mean, we shift the centroid to the position of mean of respective cluster

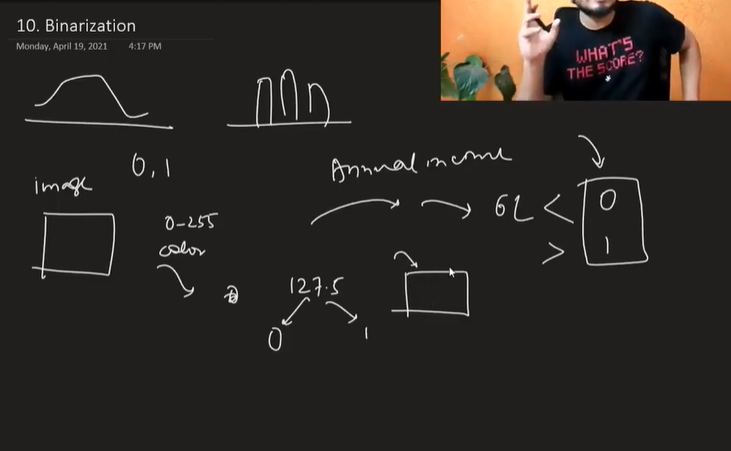
Now, we perform the same steps of clustering and shifting the centroids until when the difference b/w the original position and the new position is the same

The final position of centroid is the bins and intervals



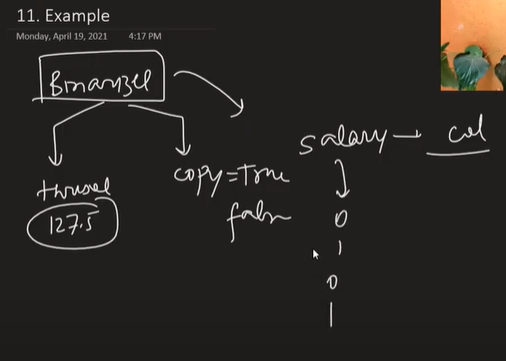


Custom/Domain Binning is performed manually based on domain expertise. We can use pandas pd.cut for this



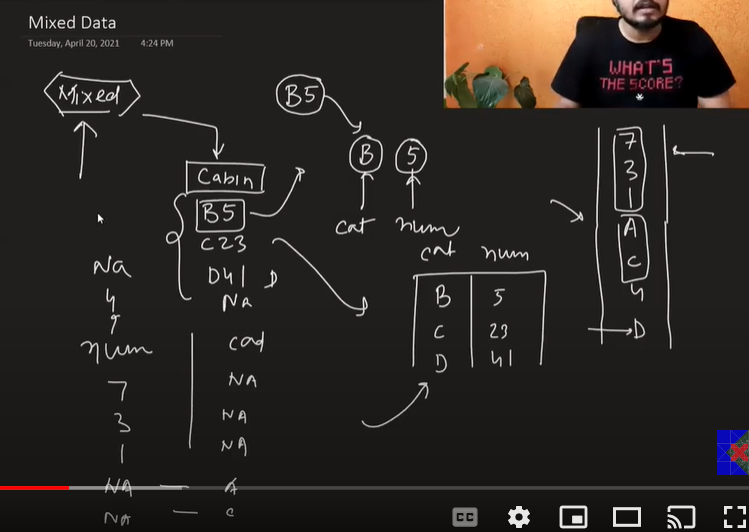
In Binarization, we split the continuous values into either 0 or 1

It is mostly used in image processing

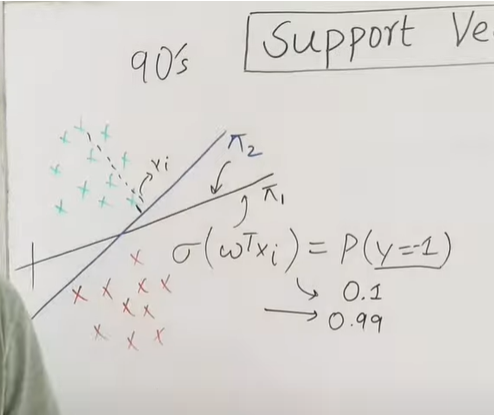


# Handling Mixed Variables | Feature Engineering

Handling a column that contains mixed datatype values can be a challenge. If there are such scenarios, we split the column into two separate columns



# Support Vector Machines | Geometric Intuition



Support vector machine is extension of logistic regression

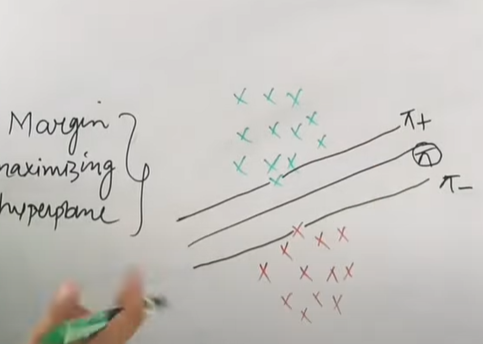
Suppose there are two lines pi1 and pi2 and they both can 100% correctly classify both classes

Logistic Regression could easily consider either one of them, but not SVM. It will select pi2 as the best hyperplane/line to classify the points

What SVM does is that it wants to create more gap b/w the both classes such that it can accurately classify

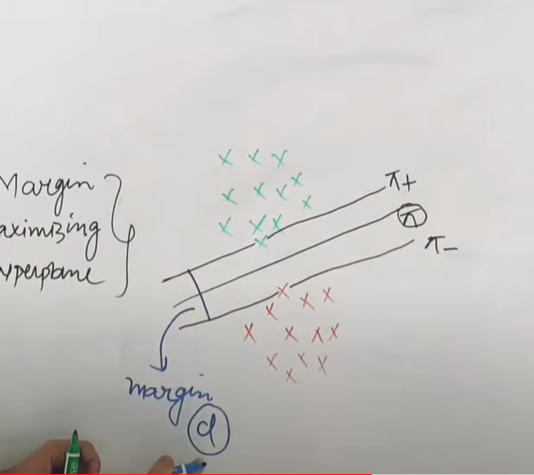
So we look for the plane that is maximizing the gap b/w the hyperplane and the classes

SVM tries to maximize the difference b/w two classes using two more hyperplanes



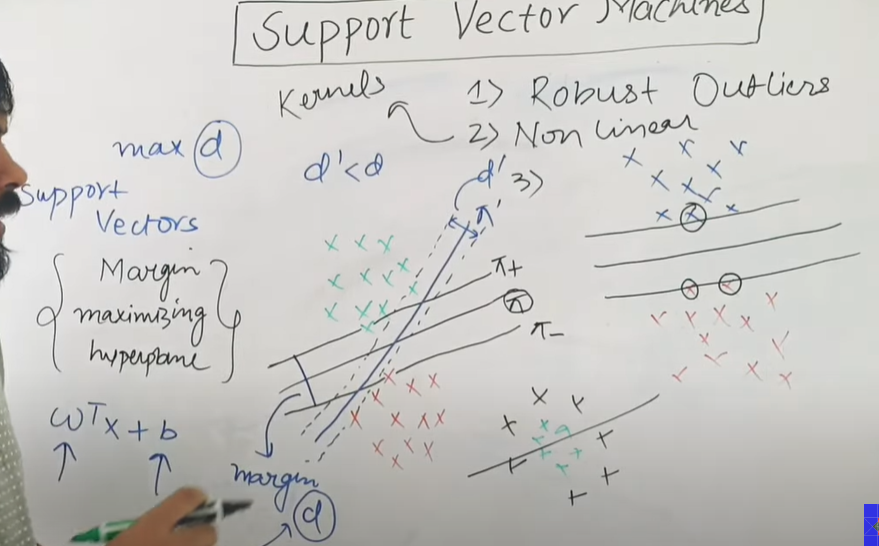
Now we draw two hyperplanes one touching the first point of positive class (pi+) and first point of the negative class (pi-)

Now we find the margin(d), shortest distance b/w pi+ and pi-



Now, we do this similar steps for bunch of other hyperplanes

Then we select that hyperplane for which the value of d is maximum

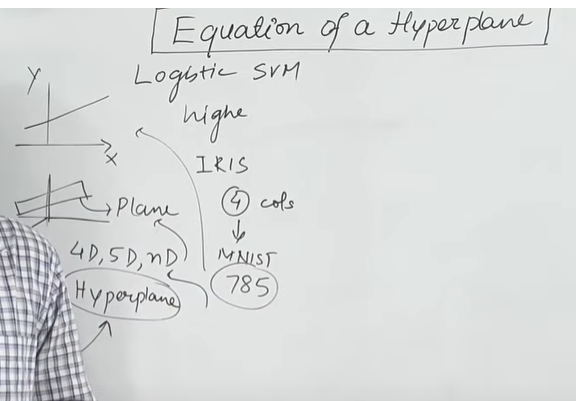


SVM will select line pi over pi’ as d is maximum

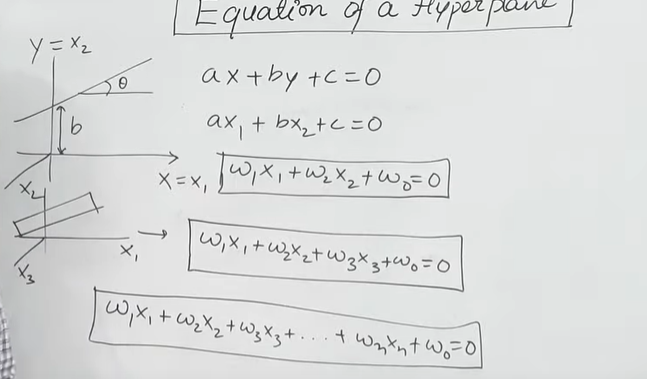
The points from each class that are touching the line are known as support vectors

Advantage of support vectors:

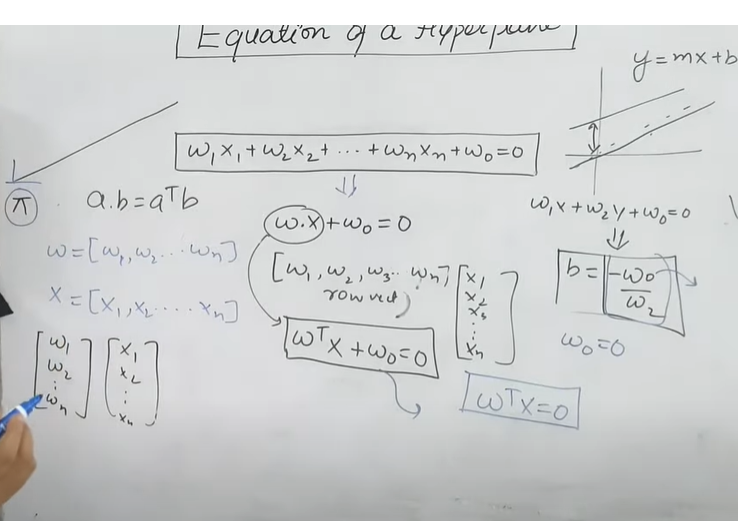
1. Robust to outliers
2. Can work on non-linear datasets
3. Can work on both classification and regression problems



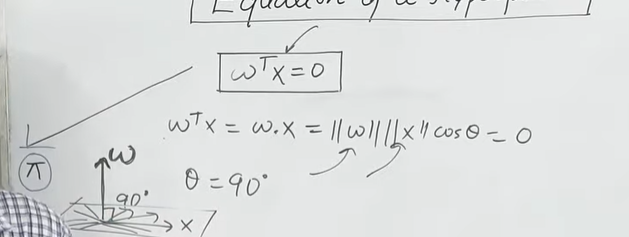
In higher dimensions, line becomes a hyperplane



Now we can normalize this equation in terms of vector algebra

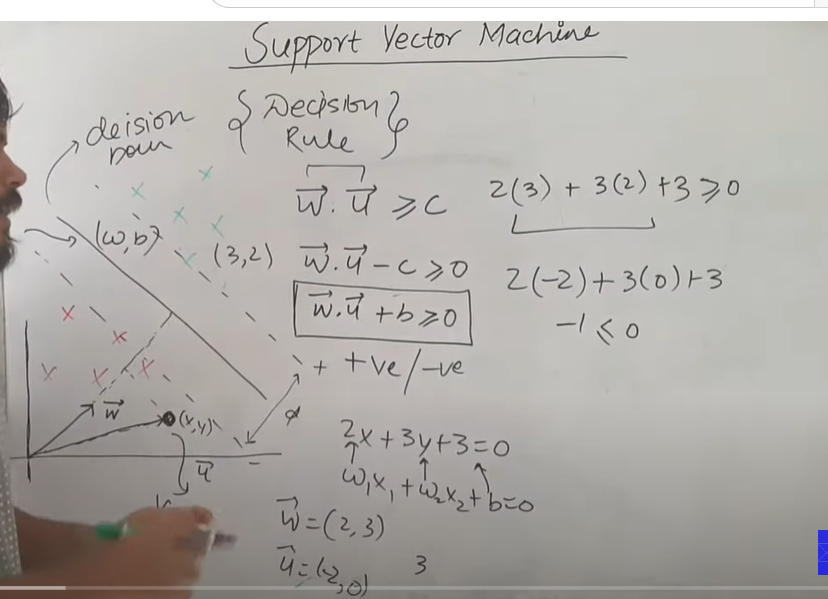


We are assuming that hyperplane is passing through the origin, so intercept coefficient becomes zero (wo)

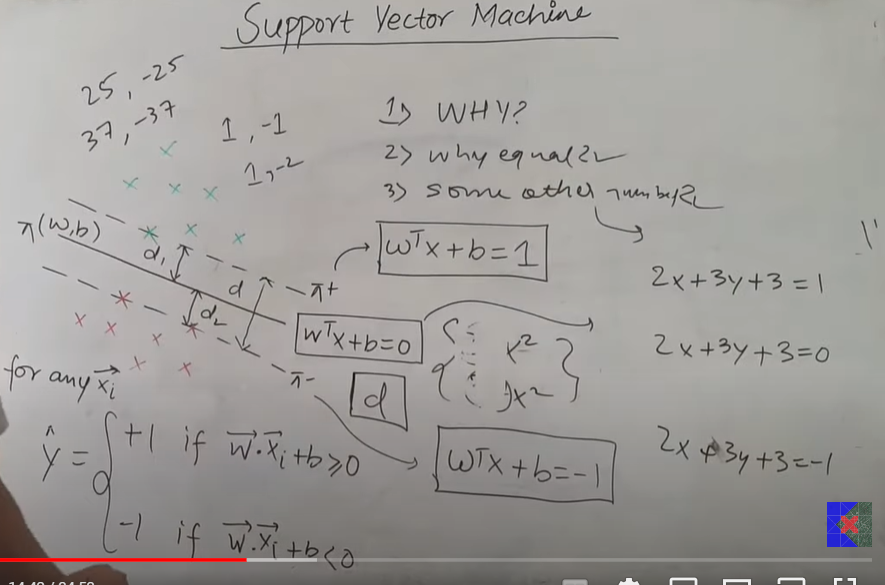


Coefficients are always perpendicular to x (independent features)

# Mathematics of SVM | Support Vector Machines | Hard margin SVM

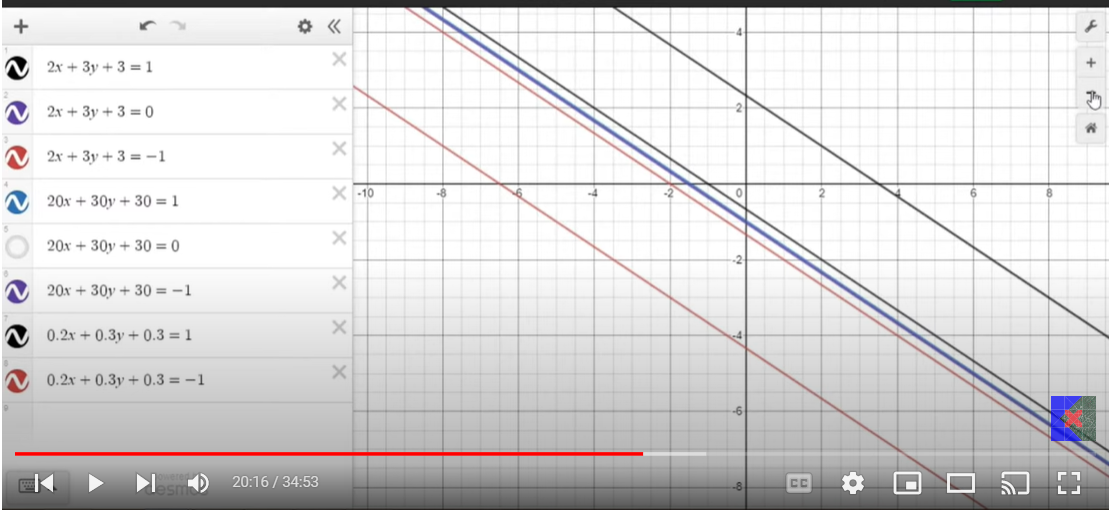


Suppose we know w vector with the coefficients, and we need to classify new points in either positive or negative region. We consider the unit vector (x,y) for the new point to be classified and project it to w vector and then calculate the dot product of u and w vectors ( since we are projecting). If u.w > c (distance from threshold) or u.w +b > 0 then we classify it as positive point and if < 0 then negative point

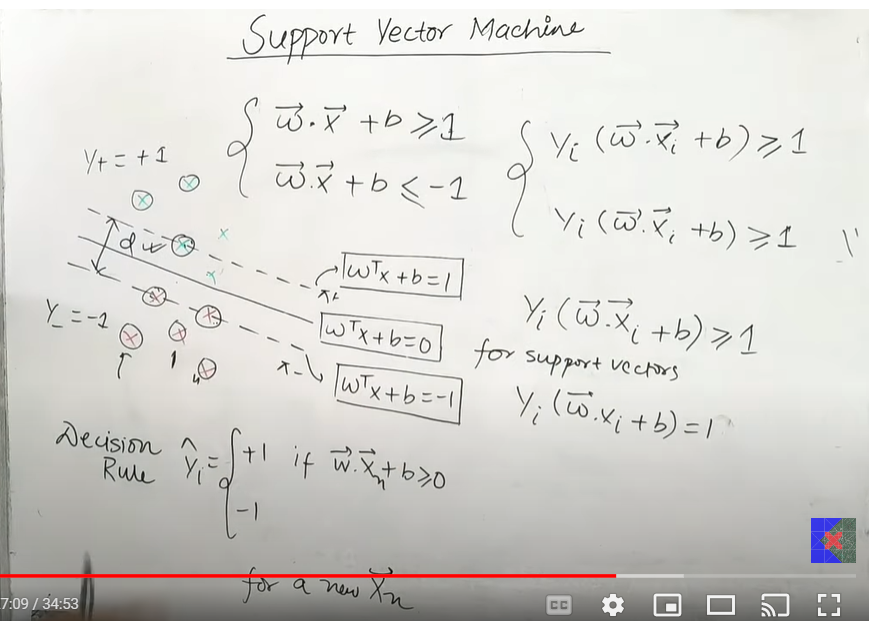


Now our aim is to find the equation of hyperplane, for that we need to find d b/w pi1 and pi2

We have to make sure that pi is at equal distance from pi+ and pi-



We can observe that changing the coefficients doesn’t make difference on the hyperplane but increasing the coefficients (greater than 1) shifts the hyperplane boundary closer and decreasing coefficients (lesser than 1) shifts the hyperplane boundary way far

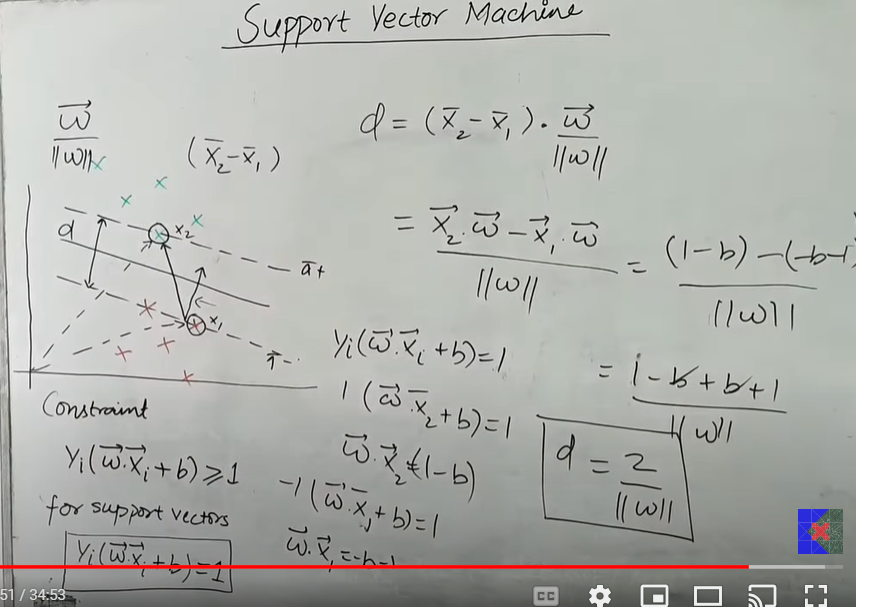


Now we need to maximize the distance d, providing all the existing points are on their same place.

So, we define a constraint that the existing point should follow wx +b >=1 and wx + b <= 1. Such that all the points are either less than or greater than decision hyperplanes and some of the points are exactly on the line (support vectors)

We can generalize both the equation into a single equation by multiplying it with respective y values (- 1 and 1) and we get the same equation

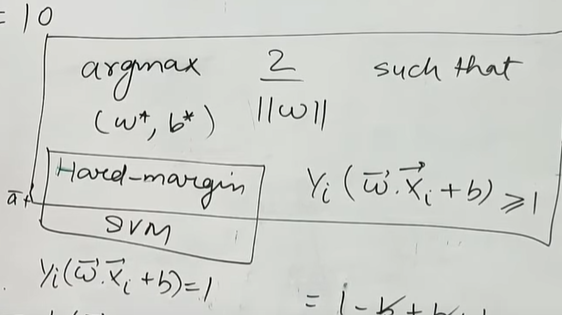
Y(wx+b)>=1, which is y(wx+b)=1 for the support vectors



Now to calculate the distance d, we find the distance b/w x1 and x2 and project it on the vector which is perpendicular to x1. So in essence, we need to dot product (x2-x1) and unit vector of w(which is perpendicular to x)

We calculate the value of x2w and x1w from the equation y(wx+b) =1

And we get value of d as 2/ |w|



This will only work if the data is perfectly within the boundary with no noise (y(wixi + b ) > 1), This is known as hard margin SVM