

LECTURE NOTES

Table of Contents

- | |
|---|
| 1. Clustering- definition, applications |
| 2. Dunn Index |
| 3. Lloyd's Algorithm |
| 4. Within Cluster Sum of Squares (WCSS), Silhouette Score, Elbow Method |
| 5. Drawbacks of the K-Means clustering, K-Means ++ |
| 6. Hierarchical clustering |
| 7. DBSCAN |
| 8. Soft clustering with Gaussian Mixture Models |
| 9. Anomaly Detection - Distribution Based |
| 10. Random Sample Consensus (RANSAC) |
| 11. Elliptic Envelope |
| 12. Isolation Forests |
| 13. Local Outlier Factor (LOF) |
| 14. High-Dimensional Visualisation - PCA |
| 15. t-SNE |
| 16. Recommender Systems: definition, Applications, Types. |

17. Market Basket Analysis, Apriori Algorithm, Association Rules
18. Content-Based recommender system
19. Collaborative Filtering system - Item-based, User-based, User-item
20. Matrix Factorization
21. How to evaluate the performance of recommender systems - Qualitative, Quantitative
22. Time Series Analysis

Clustering

- The process of grouping any kind of data based on the similarity in their features, automatically, without human expertise, is called **clustering**. It is a type of **unsupervised learning**.
- Intuitively, clustering is dividing a population into groups such that the points in one group are similar to each other. Each group is called a **cluster**.
- Since there is no ground truth data and nothing to compare with, we decide if a cluster is good or bad if it simply makes some **business sense**.

Some advanced applications of clustering:

- Validation of domain knowledge/human opinion.
- Search algorithms and recommendation systems.
- Feature creation.
- Clustering with supervised learning to improve model accuracy.

Dunn Index

- **Dunn Index** is a metric for the evaluation of clustering algorithms. It is calculated as a ratio of the **smallest** inter-cluster distance to the **largest** intra-cluster distance.

i.e.
$$D = \frac{\text{minimum inter-cluster distance}}{\text{maximum intra-cluster distance}}$$

- A high Dunn Index means better clustering since observations in each cluster are closer together, while clusters themselves are further away from each other.
- Dunn Index is **unbound**, so it can only be interpreted in a relative sense.

Best centers

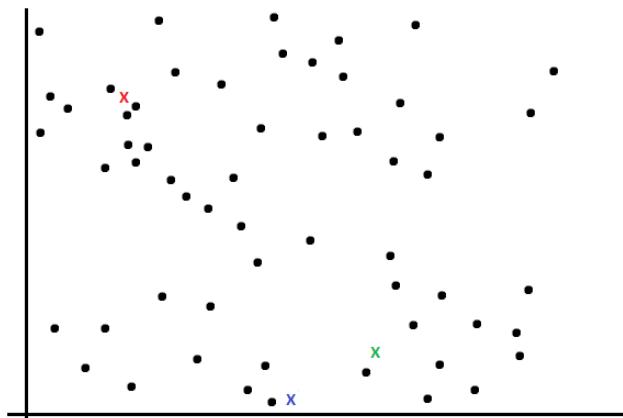
- After each iteration, the value of the Dunn Index **increases**. It is a sign that the clusters have become better after each iteration.
- The **objective** for the optimization of clusters can be represented as:
$$L : \max_c dunn(x, argmin(c, x))$$
- The above equation can be read as:
 - Take a point x and calculate its distance from all the **centers**,
 - Choose the center having minimum distance from the point, that would be the cluster in which the point x would be classified,
 - Using the cluster and the data point, calculate the Dunn index and maximize that over the centers.
- But there are some **problems** while performing the above optimization steps:
 1. The objective function is not differentiable.
 2. Assigning a point x to a group is a discrete problem, and there is no good way to convert the expression into a function that is continuous and differentiable.

- Taking the above problems into consideration, we cannot use calculus for the purpose of Optimization.

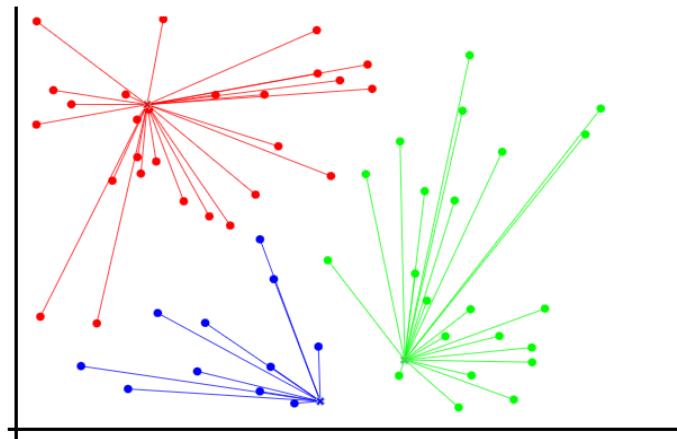
Lloyd's algorithm (K-means algorithm)

- This algorithm is used to cope with the problem of updating the centers.
- It has 5 basic steps:

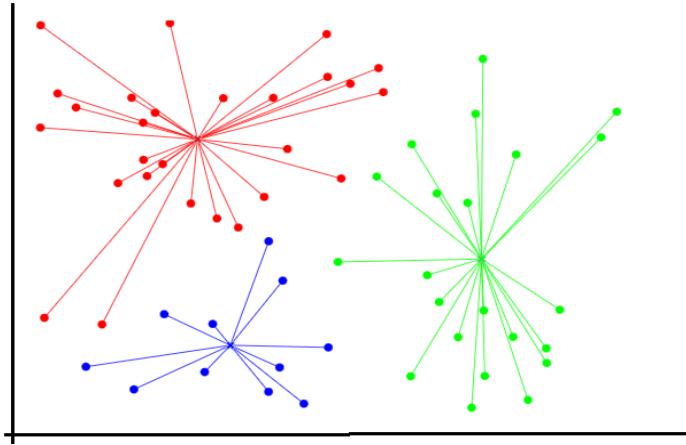
→ Randomly initialize **k** centers.



→ Assign points to these centers to get the clusters.



→ Find the **centroids** of these clusters.



- Reassign the points to the centers to create the new clusters.
- Repeat until the center of new clusters stops changing their positions.

Within-cluster sum of squares (WCSS)

- The within-cluster sum of squares is a measure of the variability of the data points within each cluster. It is given as:

$$WCSS = \sum_{i=1}^k \sum_{j=1}^{m_i} (x_{ij} - c_i)^2$$

where x_{ij} is the j^{th} point belonging to the i^{th} cluster and m_i is the number of points in the i^{th} cluster.

- A variation of the above formula can be as follows:

$$WCSS = \sum_{i=1}^k \sum_{j=1}^{m_i} d(x_{ij}, c_i)$$

where, $d(x_{ij}, c_i)$ is representing a distance function that is calculating the distance between the point x_{ij} and the centroid c_i of the cluster.

Silhouette score

- The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation).

$$S(x_i) = \frac{b - a}{\max(b, a)}$$

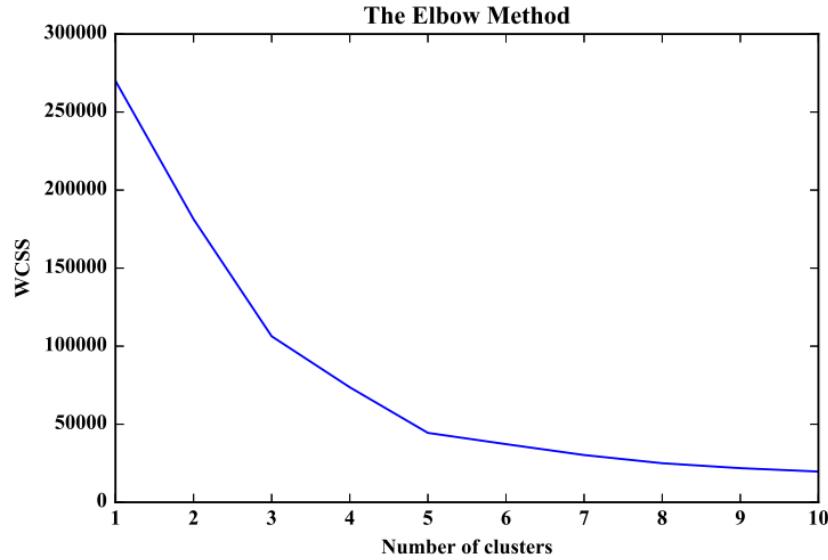
where, **a** = average distance of point x_i with points in its own cluster and,

b = average distance of point x_i with all the points of the nearest cluster.

- The range of the Silhouette score is **[-1, 1]**.
 - A Silhouette score near +1 indicates that the sample is far away from its neighboring cluster.
 - 0 Silhouette score indicates that the sample is very close to the decision boundary separating two neighboring clusters.
 - A Silhouette score of -1 indicates that the samples have been assigned to the wrong clusters.

Elbow method

- It is a method to determine the optimal number of clusters (**k**) for k-means clustering.
- We perform the k-means clustering for a range of values of **k** and for each iteration, we calculate the **WCSS** value of the WCSS metric.
- When the value of WCSS is plotted against a range of **k** values, we get a plot looking like an elbow.

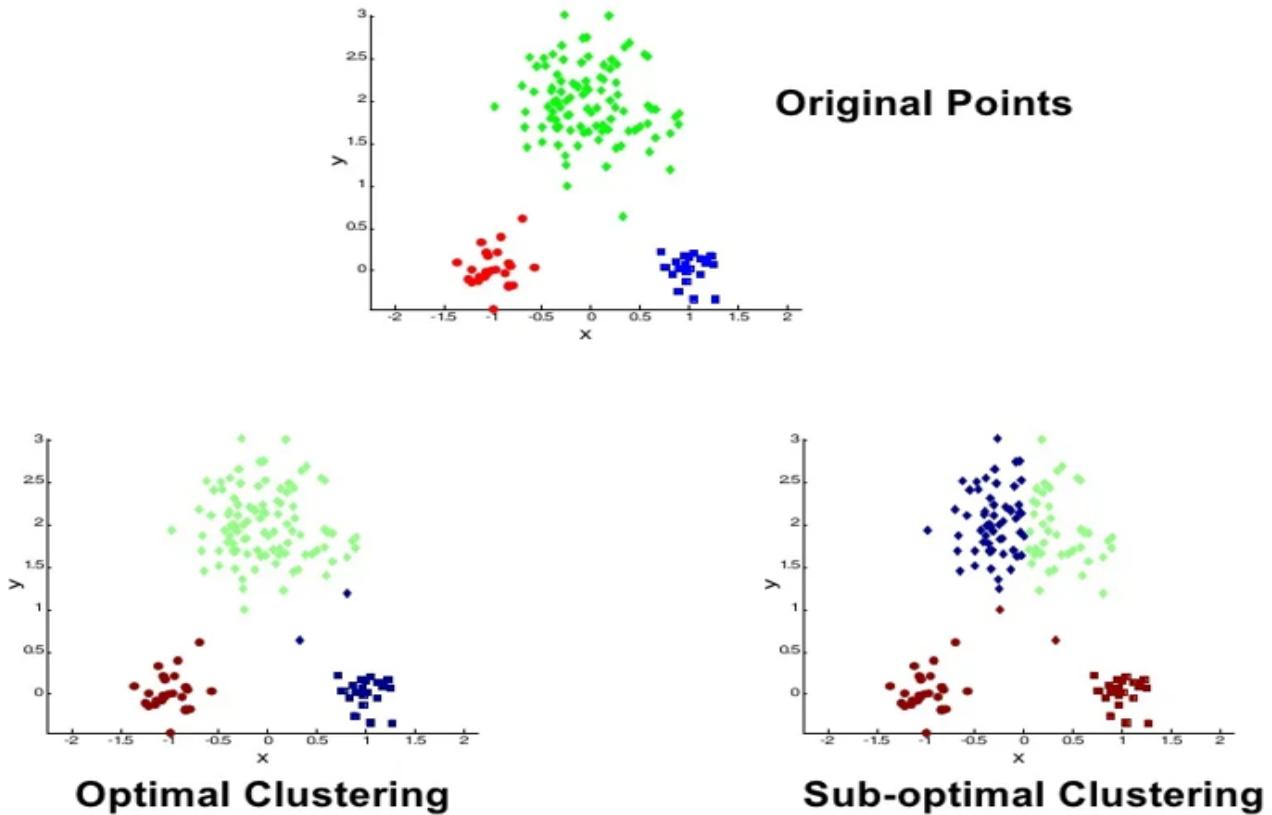


- We can clearly see that the WCSS value decreases as the number of clusters (k) increases.
- At some point on the graph, there is a sharp change in the slope ($k = 5$) after which the change in slope is very small. The k value corresponding to this point is the optimal K value or an **optimal** number of clusters.
- If we do not get a sharp change in the slope of the elbow plot while using the WCSS metric on the y-axis, we can try using the **Silhouette score** to get significant results or to get confidence in our decision.

Drawbacks of the K-means

1. We may get bad results sometimes while using the k-means algorithm because of the **random initialization** of the centroids.

For example



2. The k-means algorithm may not give the best results for data where the clusters are of varying size or density.
3. The number of clusters (k) is not defined.
4. It does not work well with non-globular clusters.

K-means++ algorithm

- To overcome the drawback due to the random initialization of centroids in K-means clustering, we use K-means++. It is smarter to initialize the centroids in order to improve the clustering algorithm.

- The steps involved in the initialization of centroids are:
 - Select the first centroid randomly from the data points.
 - Choose the next center as the farthest point from the first center.
 - The next center would be a data point farthest from both the first and second centers.
- Repeat steps 2 and 3 until **k** centroids have been sampled.
- If there are **outliers** in our data, then instead of choosing them as centroid, we can choose the farthest point as the centroid with a **probability proportional to the distance**. This is the implementation that sklearn follows by default..

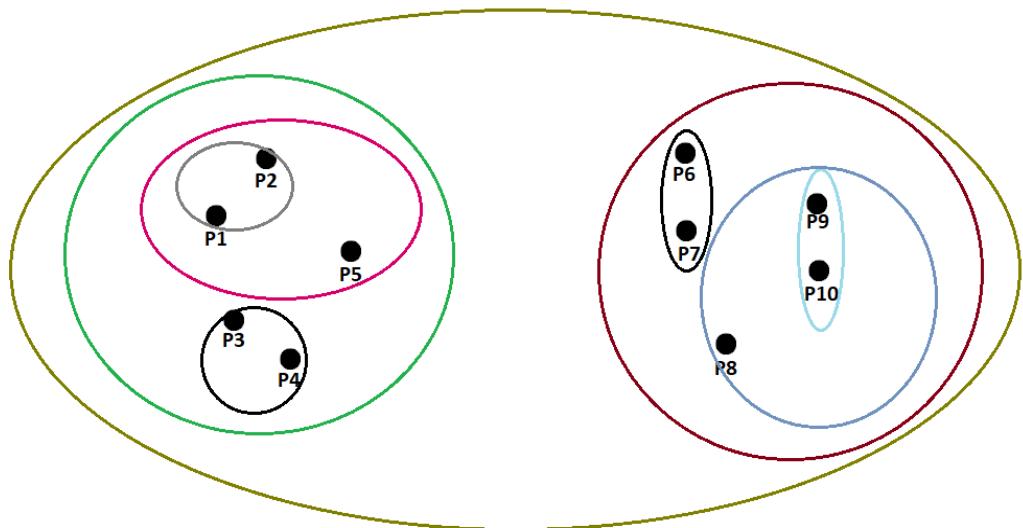
Hierarchical clustering

- It is an unsupervised clustering algorithm in which the hierarchy of clusters is developed in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.

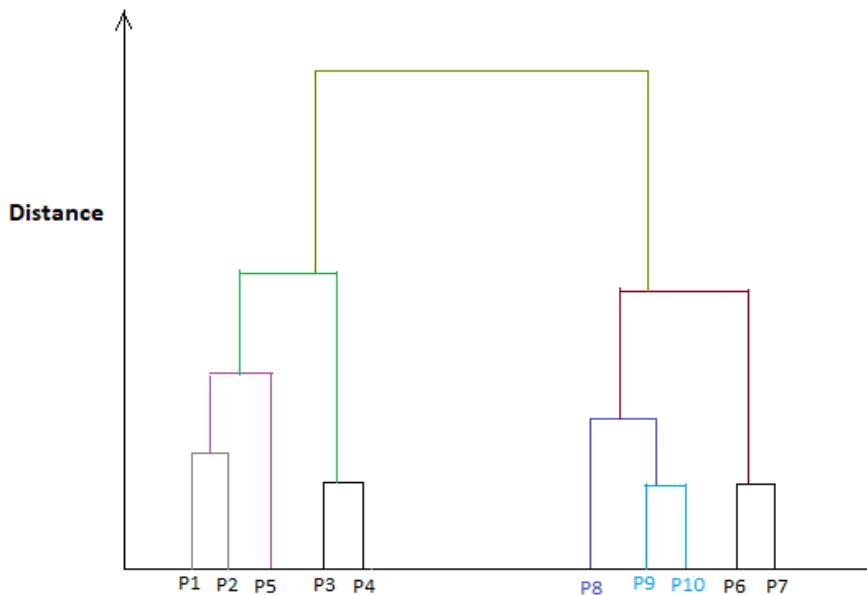
Steps of Hierarchical clustering:

- Compute the distance between all the data points. We can choose any distance measure from Euclidean, Manhattan, Hamming Distance, etc
- Merge the two points having minimum distance between them.
- Treat the sub-clusters as points and repeat till only one cluster remains.
- Finally, develop the dendrogram to divide the clusters as per the problem.

For example,



The associated **dendrogram** is:



Distance between two clusters:

- We observe that the distance between two clusters needs to be calculated in order to merge two clusters. Therefore, some linkage needs to be formed between these clusters.
- The common type of linkages that can be formed between the two clusters are:
 1. Complete linkage: It is the **maximum** distance between the two farthest points of different clusters.
 2. Single linkage: It is the **minimum** distance between the two closest points of different clusters.
 3. Centroid linkage: The distance between the centroids of the clusters is calculated.
 4. Ward's distance/linkage: The Ward's distance between two clusters c_1 and c_2 is given as:

$$W(c_1 + c_2) = WCSS(c_1 + c_2) - WCSS(c_1) - WCSS(c_2)$$

where WCSS is the **within-cluster sum of squares**.

Advantages of Hierarchical clustering:

- There is no need to pre-decide the k value. i.e. no. of clusters.
- Good for data visualization providing hierarchical relation between the clusters. The hierarchy may actually be present in the real world.

Disadvantages of Hierarchical clustering:

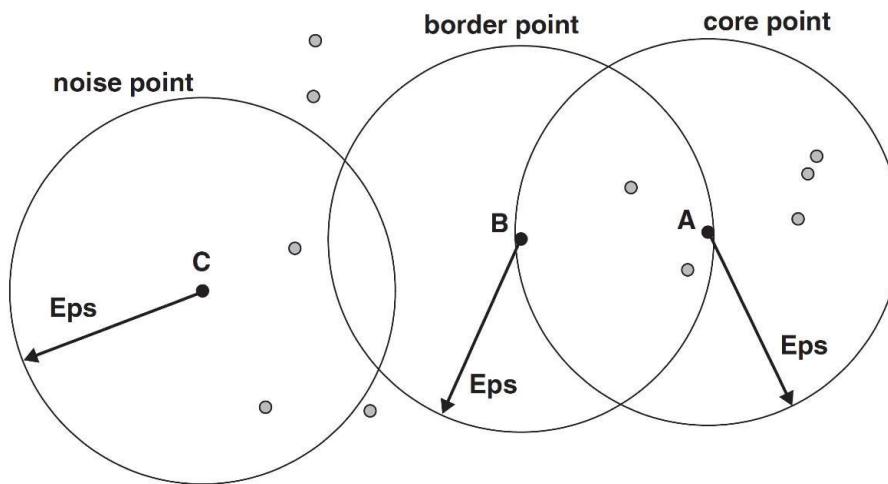
- This method of clustering is sensitive to the choice of linkage function.
- Hierarchical clustering can only be used for analysis. We cannot do inference using it. If we get a new data point, we need to re-run the entire algorithm with all the existing points and reanalyze.

DBSCAN (Density-based spatial clustering application with noise)

- The fundamental principle is that each point in a cluster must have at least a certain number of neighbors within a given radius.

- The density threshold is defined by two parameters: the radius of the neighborhood (**eps**) and the minimum number of neighbors (**minPts**) within the radius of the neighborhood.
- There are three types of points in our data:
 - 1. Core point:** This is a point that has at least *minPts* points within distance *eps* from itself.
 - 2. Border point:** This is a point that has at least one Core point at a distance *eps*.
 - 3. Noise point:** This is a point that is neither a Core nor a Border. And it has less than *minPts* points within distance *eps* from itself.

For e.g., with a value **minPts= 7**,



- The DBSCAN clustering algorithm can be summarized in the following steps:
 - Pick a random point from the dataset and if it is a **core point**, assign it to cluster 1 and mark it as visited.
 - If not, the point is marked as noise. Assign all the points within the neighborhood of the initial point to cluster 1. If these new points are also core points, assign their neighborhood point to cluster 1.

- Next, randomly choose another point that has not been visited and repeat step 1 with it.
- Stop when all the points are visited.

Deciding *epsilon*:

- One way to estimate the initial value of epsilon is:
 - Calculate the distance between each point.
 - Plot the histogram of those distances.
 - We may get two peaks, the initial value of epsilon may be one of the distance values between these two peaks.

Advantages of DBSCAN:

- It is able to find arbitrarily shaped clusters and clusters with noise (i.e. outliers).
- There is no need to decide 'k' (no. of clusters).

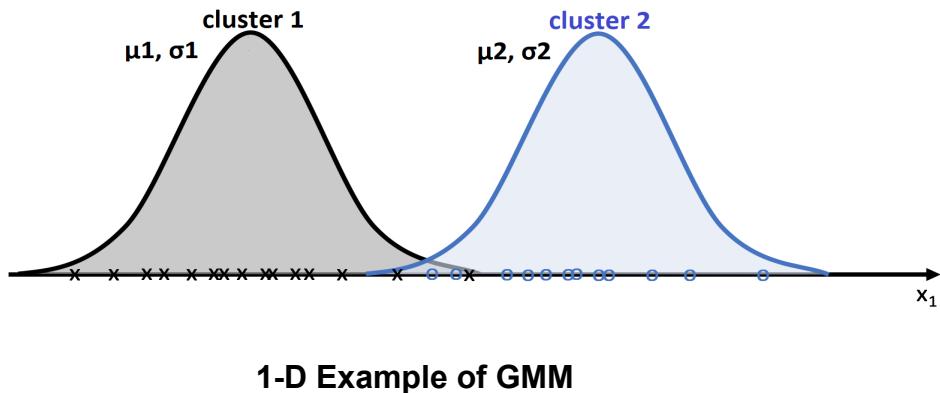
Disadvantages of DBSCAN:

- It does not work well with sparse points (high dimensional data).
- While DBSCAN is great at separating high-density clusters from low-density clusters, DBSCAN struggles with clusters of similar density.

Soft clustering with Gaussian mixture model

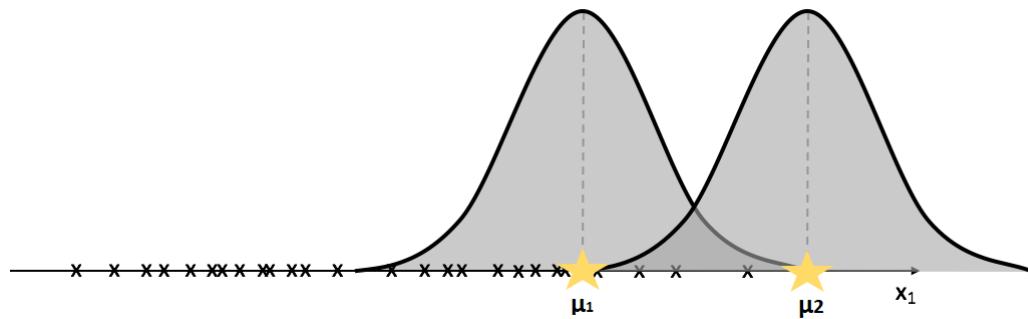
- In soft clustering, a probability of each data point being in different clusters is assigned instead of putting each data point completely into a separate cluster.
- Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters.
- We can represent the clusters in our data using the **Gaussian distributions**.

- For example, in the below given 1-Dimensional data, each cluster is represented using a Gaussian distribution.



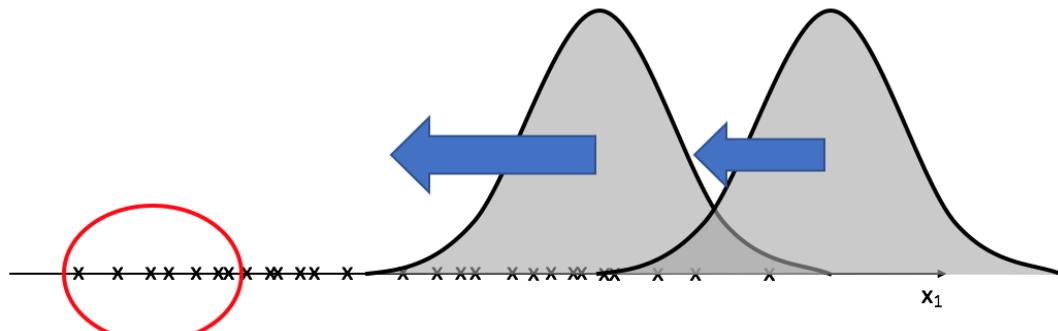
GMM algorithm:

→ Randomly initialize μ and σ .



→ Update μ and σ .

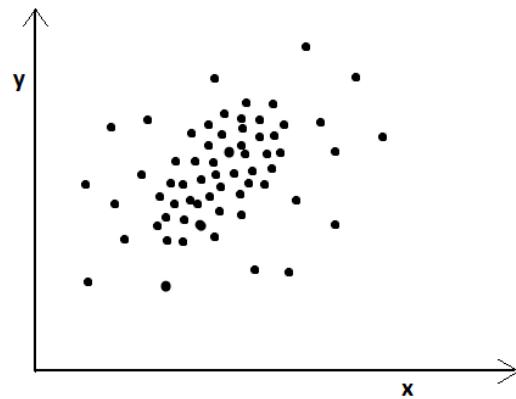
- Calc the probability of each point belonging to each gaussian.
- Assign all the points which have the highest probability to their respective Gaussians.
- Now, compute new μ for each gaussian as the mean of the points belonging to that gaussian (similar to Lloyd's algorithm)
- Similarly, update the variance.



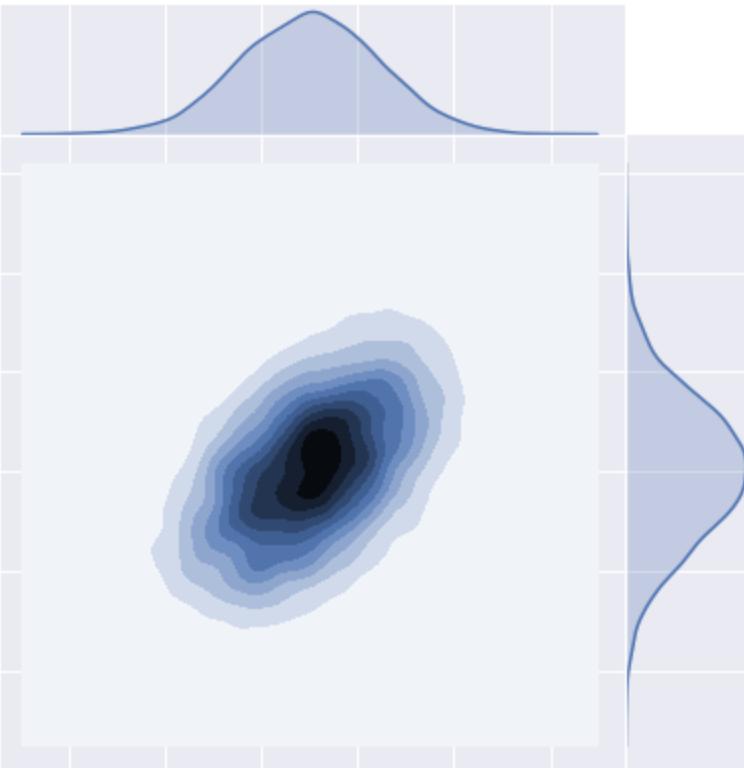
→ After multiple updates, we will have tightly fitting Gaussian distributions representing different clusters.

2-Dimensional Gaussians:

- Let's consider two variables x and y with the following 2D scatter plot.



- The corresponding Multivariate Gaussian distribution would be:



- The part where the color is dark has more density of points and vice versa.
- The above multivariate Gaussian distribution is defined by five parameters:
 1. μ_x - the mean of x-coordinate.
 2. μ_y - mean of y-coordinate
 3. σ_x^2 - variance in x direction.
 4. σ_y^2 - variance in the y direction.
 5. $\text{Cov}(x,y)$ - Covariance between x and y.
- The above ideas can be applied to 2D and ND gaussian, and this is how the GMM algorithm works.

Anomaly Detection

So far, you have learned many concepts that are there in Machine Learning

The purpose of this lecture is to let you know how you can take the concepts that you already know like Random Forests, SVMs, KNN, etc., and modify them for using them for another purpose rather than classification.

What is an Anomaly?

- Anomaly is synonymous with an outlier. These terms are often interchanged and may be called Novelty depending on the context.

What's the difference?

- Anomaly means something which is not a part of the normal behavior
- Novelty means something unique, or something that you haven't seen it before(novel)

Applications of Anomaly/Outlier/Novelty Detection:

- Credit Card Fraud Detection
- System Intrusion Detection
- Ecosystem Disturbances in Weather and Environment (alarming about Tsunami, Earthquake, etc.)
- You can read how some startups are thinking out of the box using these techniques from a blog [here](#).

1. Distribution Based

- The simplest way for detecting an outlier would be to use distribution parameters (mean and standard deviation).
- Consider a feature X with some observations x_1, x_2, \dots, x_n with some outliers.
- We know that it will follow some distribution which will have parameters θ .
- Let's follow a gaussian distribution with some mean(μ) and standard deviation(σ)
- If we know the distribution of the data, we'll try to fit the distribution. But, the problem arises with the distribution parameters.
- While we know the distribution, the parameter estimates of the distribution are often corrupted by the noise/outlier
- Hence, we need to robustly estimate the parameters of the distribution. We do this using an algorithm called RANSAC which stands for Random Sample Consensus

2. Random Sample Consensus (RANSAC)

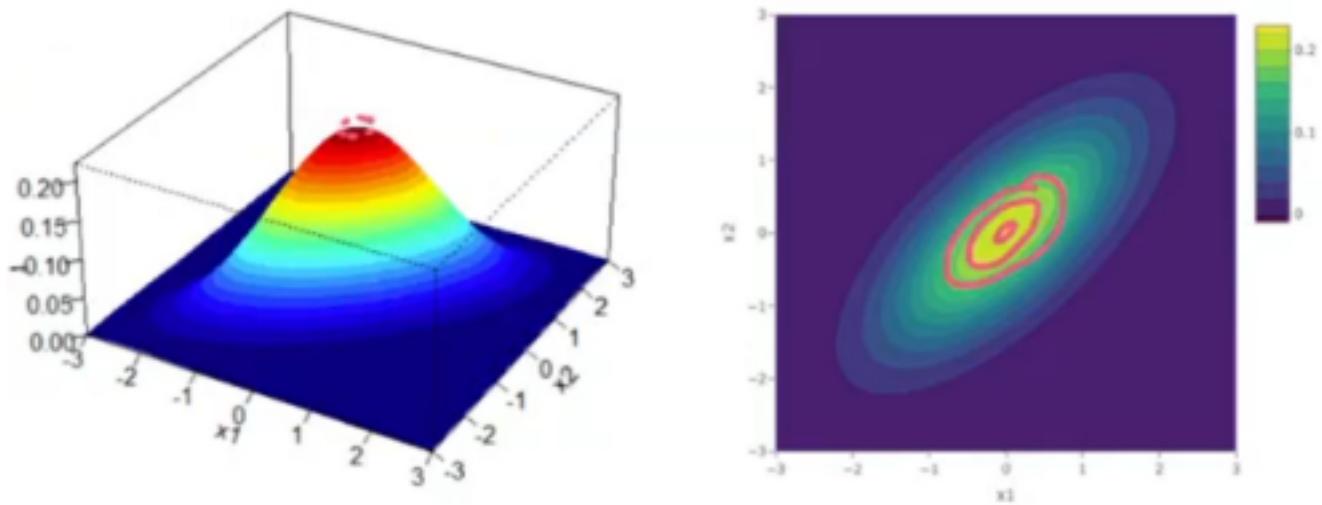
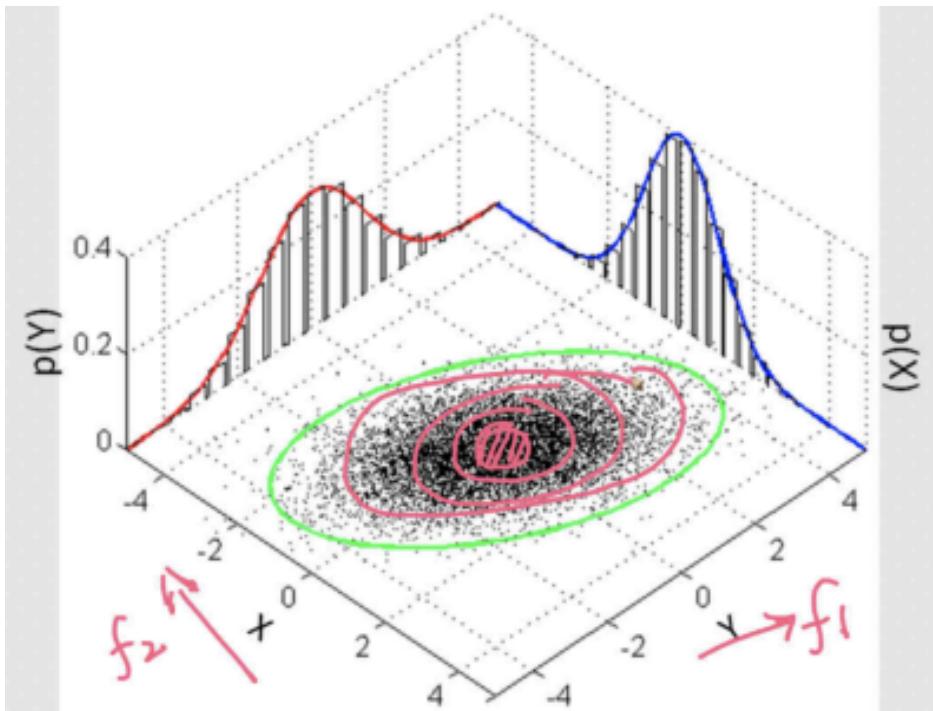
- RANSAC is a trial-and-error approach that works very well in real life.
- Imagine a dataset X with a number of points having parameters μ and σ^2 . Let's call them collectively θ
- There are mainly three steps involved in RANSAC. They are
 - Sample a subset of points from the dataset (n'). We consider this point as an inlier.
 - Now, compute a model that estimates the parameters of the sampled points.
 - Score the model which indicates how many points will support the model.
- We repeat these three steps iteratively and then select the model best supported by the data, which then tells which points are inliers and which are outliers.

Extending the idea to higher dimensions

- Till now, we assumed that we have only a single feature X which was following Gaussian Distribution.
- Now, imagine we have d -dimensional data where each point $x_i \in \mathbb{R}^d$ and the data is not labeled.
- If we know the data points x_i s follow multivariate gaussian distribution(unimodal), then X follows normal distribution; $X \sim (\vec{u}, \Sigma)$, where \vec{u} is mean vector and Σ is a covariance matrix
- Here we'll consider (\vec{u}, Σ) as θ
- If you remember from GMMS, in multidimensional space, the shape of gaussian was similar to a hill where the density of the points was highest in the middle contour, and it keeps getting low as we move away from the center
- In this case too, RANSAC can be applied. Farther away from centroid, we'll know that it is an outlier.
- Similar principle is used in another method called Elliptical Envelope.

3. Elliptic Envelope

- We know that a Unimodal Multivariate Gaussian Distribution on a single plane will look like ellipses if visualized on a plane. This idea can be extended to find out an outlier



- Given some data X where $x_i \in \mathbb{R}^d$ and X follows Normal Distribution being unimodal, Elliptical Envelope robustly estimate the parameters (\vec{u}, Σ) .
- The term robustly means without getting impacted by outliers
- Next, then we remove the points that are outliers which are very far away from the centroid

What if the distribution is non-gaussian? Do we need to convert it into

gaussian distribution?

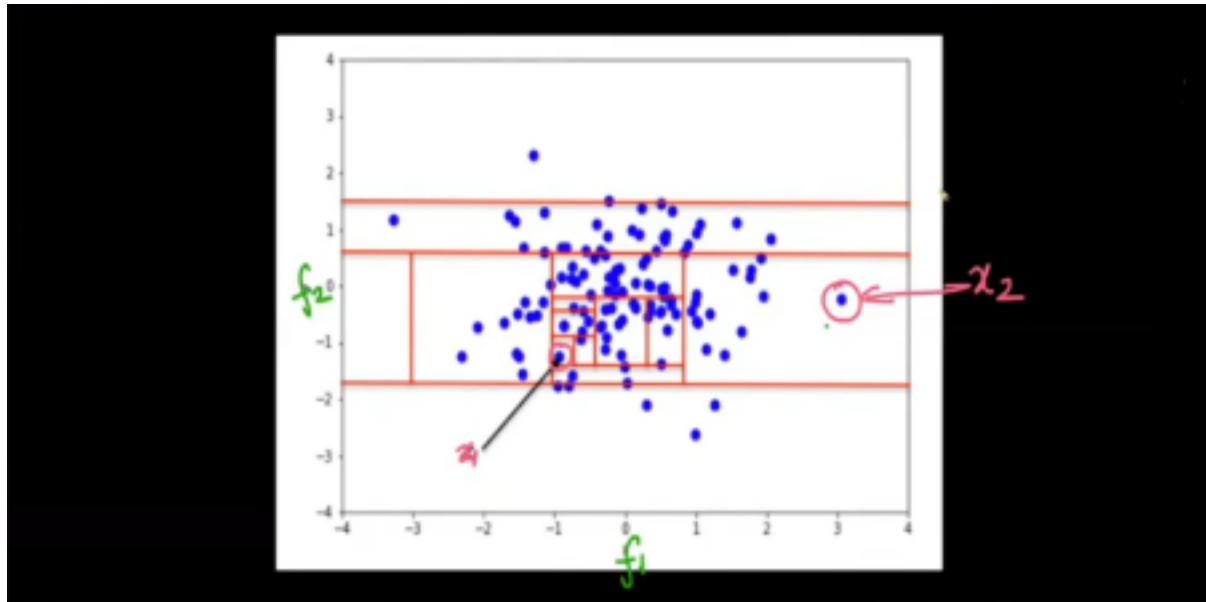
- While the elliptical envelope method makes an assumption that the distribution is gaussian, the strategy can be applied to any distribution.
- As long as we know any distribution and its parameters θ , we can extend our strategy to use RANSAC and estimate parameters θ .
- There can be other distributions such as multivariate Poissons, multivariate log normals, etc., but we don't use them as much.
- One other strategy is to convert them into Gaussians but we don't necessarily have to.
- As long as there is any distribution we can calculate the probability of $x_i \in X$ using PMF/PDF.

Disadvantages:

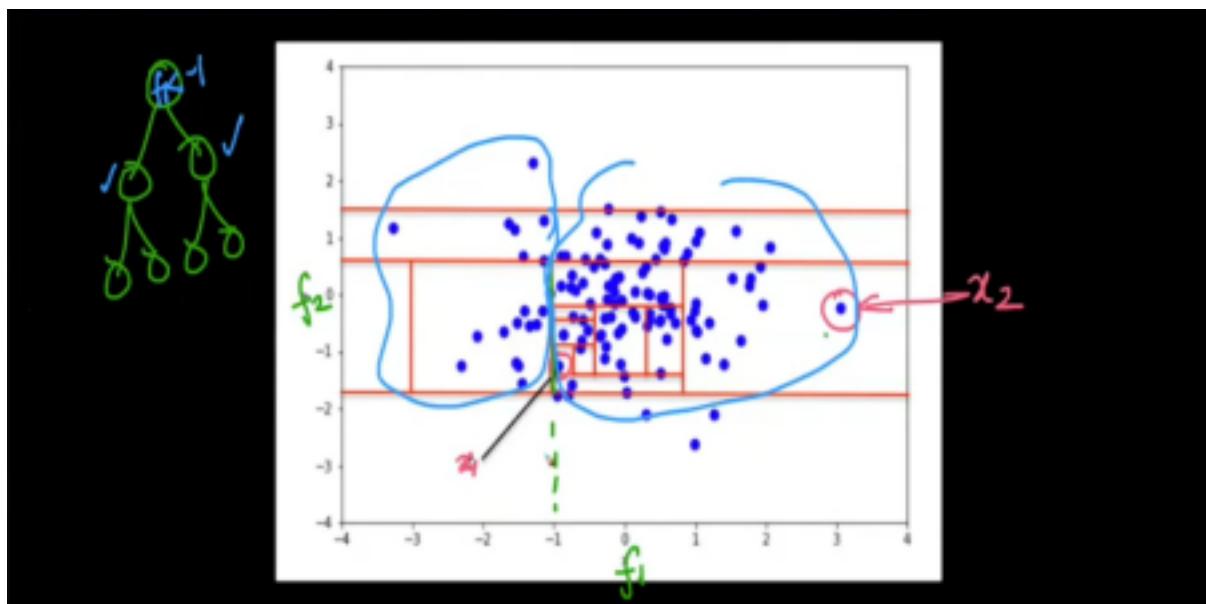
- It cannot be used for non-unimodal data
- It is specifically for multivariate Gaussians
- If the data fails to meet the assumptions of unimodal and multivariate gaussian, the whole thing crashes.

4. Isolation Forests

- Consider a dataset D which contains data points x_1, x_2, \dots, x_n . Just like Random forests, Isolation Forests build many trees.
- These are the steps involved in Isolation Forest:
 - Build many trees like random forests
 - For each tree:
 - Randomly pick a feature
 - Randomly threshold that feature
 - Build each tree until the leaf consists of only one datapoint
- Isolation Forests are also known as iForests
- Consider the plot along feature f_1 and f_2 given below:



- In isolation forests, we are building totally random trees. So if we pick feature f_1 and put a threshold there will be a vertical bar.
- Similarly, if we pick feature f_2 and put a threshold there will be a horizontal bar.
- For example, if we pick feature f_1 and we select threshold as $f_1 < 1$, then our first root node will be based on this condition



- Based on the diagram above,
 - The node containing x_1 will be at more depth.

- Observe that the point x_1 is in dense region, and point x_2 is far away
- That is because, to break the point x_1 from all the other points, more and more splits will be required and that will increase the depth of the node containing point x_1 .
- So, to sum it up, the idea behind Isolation forest is:
 - On an average outliers have lower depth in the random trees
 - On an average, inliers have more depth in the random trees

Evaluation of Isolation Forest

- Imagine, we have build random trees. For each point x_i in the dataset, we can get an average depth.
- We use this average depth to convert into a metric.
- Apart from this, there are lot of different metrics, that people have came up with over the years
- But, the basic intuition is that lesser the average depth, higher likelihood is there that it is an outlier

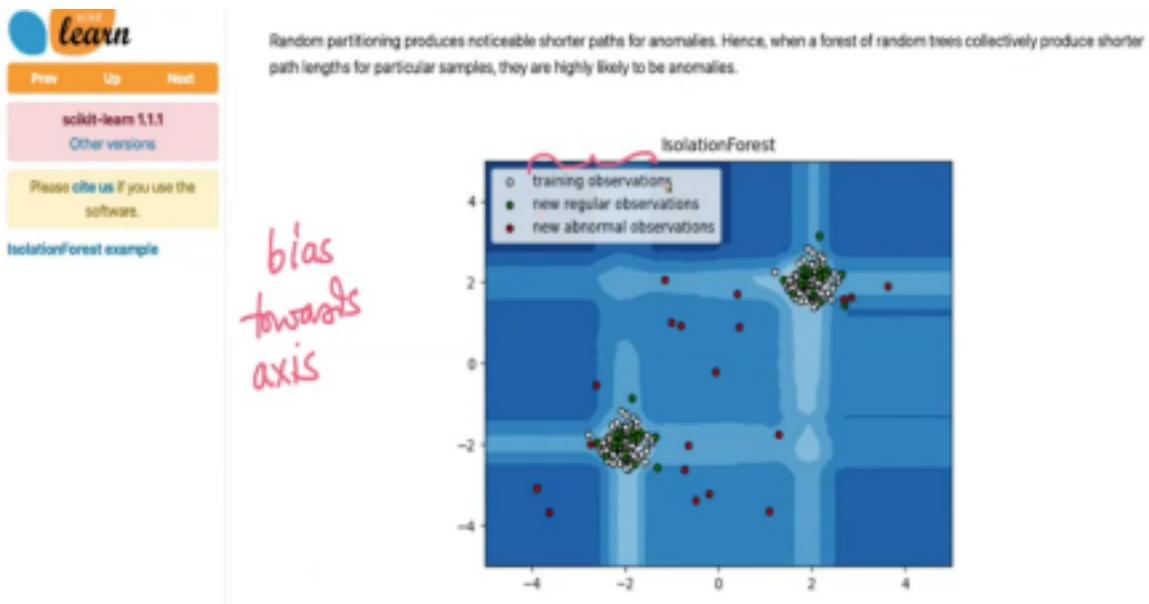
Deciding average depth of a point

- There is no one metric specifically used for average depths in iForests. At the end, whichever metric you use, it is based on the threshold.
- There are a lot of metrics that researchers have came up with over the years.
- But, studying them in this lecture is out of scope.

Disadvantages

- One of the major limitation of iForests is that they are biased towards axis parallel splits.
- iForests makes splits and these splits are always parallel to either of the axis.
- Because of this, the boundary will not be smoothed.
 - In the diagram given below, the different shades of blue represents the likelihood of a point to be an outlier. Darker the color, it is more likely that the point in that region will be an outlier

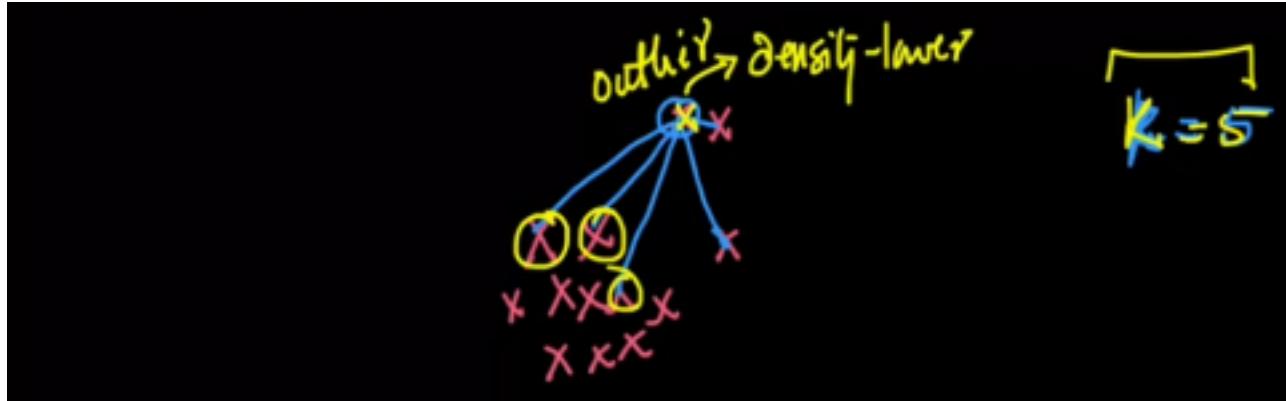
- We've trained iForest model using training data(white points)
- It is tested on testing data(red + green) where red color indicates outliers



- Now imagine two points x_1 and x_2 as shown in the diagram given below.
- Both the points are almost equidistant from nearest cluster. x_1 is on the axis and point x_2 is off axis.
- Because the model is biased towards axis, it will classify point as an inlier and as an outlier
- This is also known as banding in signal processing

5. Local Outlier Factor (LOF)

- On a higher level, LOF is based on two ideas: KNN and density
- The core idea behind LOF is to compare the density of a point with its neighbors' density
- If the density of a point is less than the density of its neighbors, we flag that point as an outlier
- Imagine a bunch of datapoints as shown below



- We compute the density of a point based on average distance.
- If average distance between a point and its K nearest neighbors is large, it is more likely that the point will be an outlier
- Also, larger the value of K , more confident are the results.

Some concepts to understand the working of LOFs:

1(a) K-distance

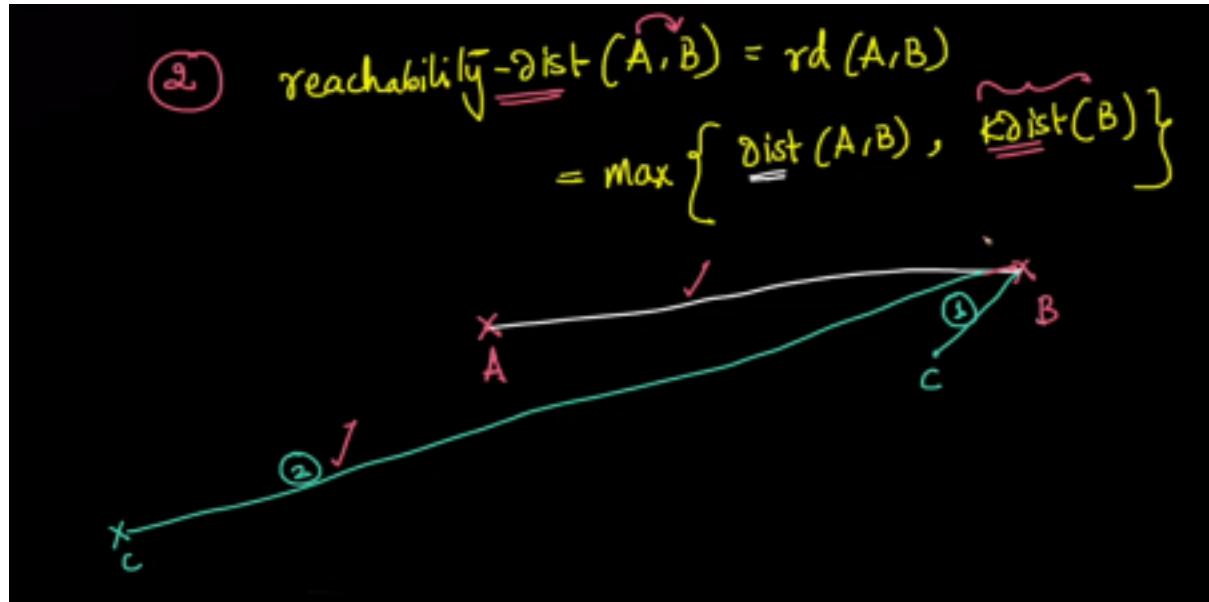
- We define K-distance of a point **A** as the distance of point **A** to its K^{th} nearest neighbor
- In general, larger the value of k-distance is, farther away the points is from other datapoints

1(b) Set: $N_k(A)$

- It is a set of k-nearest neighbors of point **A**.

2. Reachability distance

- From point **A** to point **B**, we define reachability distance as maximum of the distance from point **A** to point **B** and the maximum k-distance of point **B**
- Consider point **B** with some k nearest neighbors shown in the diagram below.



- There is a possibility that some neighbors might be close(condition 1) and some neighbors might be very far away(condition 2)
- In this case, there is a neighbor of point **B** whose k-distance is greater than the distance between point **A** and **B**, and hence, it is considered as its reachability distance.

3. Local Reachability Density

- It is often represented as $lrd_k(A)$, which tells local reachability density of a point **A**
- It is defined as the average reachability distance between point **A** and **k** neighbors

$$\text{So, } lrd_k(A) = \frac{\sum_{B \in N_k(A)} rd_k(A, B)}{N_k(A)}$$

- The summation in above equation represents the sum of reachability distances from a point **A** and set of neighbors **B** as $B \in N_k(A)$
- We define Local Outlier Factor of point as follows:

$$LOF_k(A) = \frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)|.ldr_k(A)}$$

- $ldr_k(\mathbf{A})$ is the density of point \mathbf{A}

- The expression $\frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)|}$ is the average neighborhood density
- So, LOF of point \mathbf{A} is nothing but the average neighborhood density(lrd) of point \mathbf{A} divided by the density of \mathbf{A}

Interpretation of LOF

- If $LOF(\mathbf{A}) = 1$, then we can say that the point has same density(lrd) as its k nearest neighbors
- If $LOF(\mathbf{A}) > 1$, then the k neighbors of point \mathbf{A} have higher density than point \mathbf{A} .
 - That does not mean point \mathbf{A} is an outlier. It may or may not be.
 - But if $LOF(\mathbf{A}) \gg 1$, then the point is definitely an outlier.
- If $LOF(\mathbf{A}) < 1$, then the point has more density than its nearest neighbors.

Disadvantages of LOF

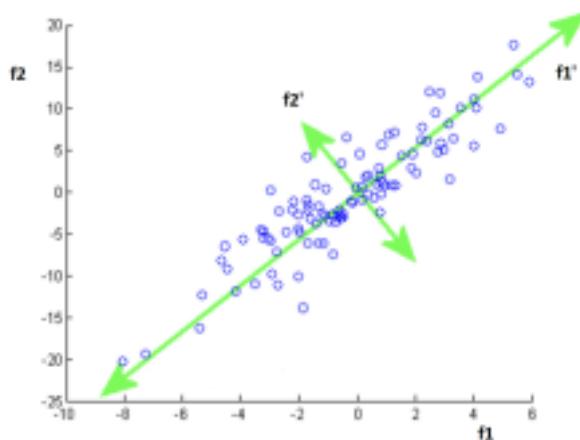
- Finding optimal K
- Finding threshold.
 - If $LOF(\mathbf{A}) \gg 1$, what is the threshold??
- Cannot handle high dimensional data efficiently
- High Time Complexity

High Dimensional Visualisation

1. Principal Component Analysis (PCA)

- Dimensionality reduction techniques help to convert high dimensional data to fewer dimensions which can be then visualized using simpler plots or it can be used when we want to preserve the information of our feature columns.
- Principal component analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets.
- PCA is an act of finding a new axis to represent the data so that a few principal components may contain the most information.
- The main objective here is whenever we are going from a higher dimension(d) to a lower dimension(d') we want to preserve those dimensions which have high variance (or high / information.)

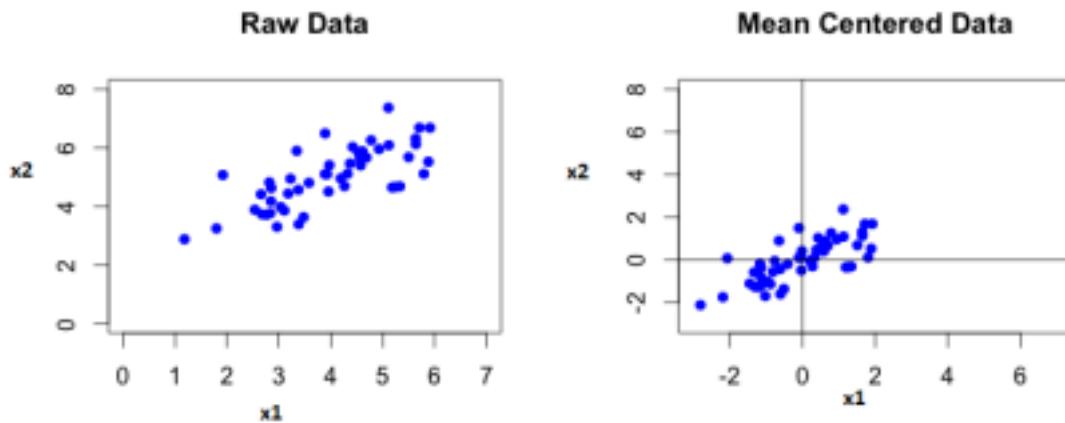
For example: Imagine we have two features f_1 and f_2 , and the data is spread as shown in the image below. We want to project the given 2-D data to 1-D.



- One thing we can observe here is that both axes have a good amount of variance. It will be difficult to decide which feature to drop.
- Therefore, we rotated the entire coordinate axes. It is shown in green.
- Let's call these new dimensions as f_1' and f_2' . Now we can see that variability on f_1' is more than f_2' and we can now drop the f_2' axis.

Mathematics of PCA:

Step 1: Feature wise Centering: We compute the mean of each feature and subtract it from each value of that feature to center the mean of the data at origin.



Step 2: Standardize: It is done in order to keep all the features of our data in same scale. Best is to use standard scaling because it is less affected by the outliers.

Combining step1 and step 2, we get the value of our feature vector x as:

$$x' = \frac{x - x.\text{mean}()}{x.\text{std}()}$$

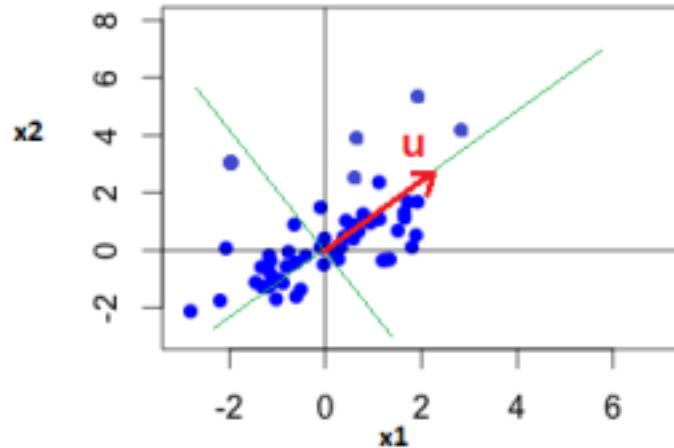
Step 3: Rotate the coordinate axes:

- There can be many angles of rotation possible, we'll consider a random axis and optimize it to get the best axis.

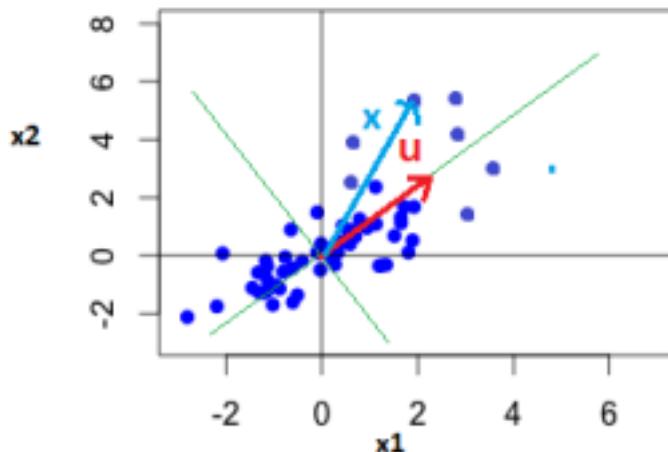


- Let's consider a unit vector in the direction of the new axis.

- Our optimization problem is finding unit vectors u such that the variance of all x_i (features) are maximum.



- We can also think of it in terms of the length of projections.
- Consider a vector x in the space representing one of the points in our data.



- The projection of a vector x on vector u is given as: $\frac{\vec{u} \cdot \vec{x}}{\|\vec{u}\|}$
- The best u will be where the summation of the length of projections of all such

points(x_i) on the vector u is maximum.

$$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^n \frac{\vec{u} \cdot \vec{x}_i}{\|\vec{u}\|}$$

i.e.

- However, the projections may be a negative value. Therefore, we take the magnitude of the numerator:

$$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^n \frac{|\vec{u} \cdot \vec{x}_i|}{\|\vec{u}\|}$$

i.e.

- Since the above objective function is not differentiable, we rewrite it as:

$$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^n \frac{(\vec{u} \cdot \vec{x}_i)^2}{\|\vec{u}\|}$$

- Now, since u is a unit vector, we introduce a constraint: $\|u\| = 1$
- Using Lagrange's multiplier, our loss function becomes:

$$\max_{\vec{u}, \lambda} \frac{(\vec{u} \cdot \vec{x})^2}{n} + \lambda(\|\vec{u}\| - 1)$$

We can optimize the above Lagrangian loss function using Gradient descent.

Optimization without Gradient descent:

- We know that, $\vec{u} \cdot \vec{x}_i = \vec{x}_i \cdot \vec{u}$

therefore, our new loss function is:

$$L = \frac{(X \cdot \vec{u})^2}{n} + \lambda(||\vec{u}|| - 1)$$

- Now, since we can write the constraint $\|\vec{u}\| = 1$ as $\|\vec{u}\|^2 = 1$, therefore,

$$L = \frac{(X \cdot \vec{u})^2}{n} + \lambda(||\vec{u}||^2 - 1)$$

- Now we know that If a matrix A is an invertible square matrix, then:

$$A^2 = A^T \cdot A = ||A||^2$$

Thus,

$$L = \frac{(X \cdot \vec{u})^T (X \cdot \vec{u})}{n} + \lambda((u^T \cdot u) - 1)$$

- Using the identity, $(A \cdot B)^T = B^T \cdot A^T$

We get,

$$L = \frac{(u^T \cdot X^T)(X \cdot \vec{u})}{n} + \lambda((u^T \cdot u) - 1)$$

$$L = u^T \frac{X^T X}{n} u + \lambda(u^T u - 1)$$

Or

$$\frac{X^T X}{n} = V$$

Let

- Hence, $L = u^T V u + \lambda(u^T u - 1)$

V is the pairwise covariance matrix of all the features of our feature matrix X .

- Taking partial derivatives w.r.t μ and λ ,

$$\frac{\partial L}{\partial u} = 2Vu + 2\lambda u \quad \text{and} \quad \frac{\partial L}{\partial \lambda} = u^T u - 1$$

- After putting both the partial derivatives equal to zero, we get,

$$Vu = \lambda u \quad \text{--- (i)}$$

$$u^T u = 1$$

- By carefully observing the equation (i), we can conclude that u is the eigenvector and λ is the eigenvalue of matrix V .

Eigenvector and Eigenvalue:

- For any matrix A , there exists a vector x such that when this vector is multiplied with the matrix A , we get a new vector in the same direction having a different magnitude.
- The vector x is called the Eigenvector and the length is called as

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$$

Eigenvector of Matrix A

Eigenvalue. i.e.

Eigenvalue of Matrix A

- There can be multiple eigenvectors, which are always orthogonal to each other.

Conclusion:

- To find the best value of vector \mathbf{u} , which is the direction of maximum variance (or maximum information) and along which we should rotate our existing coordinates, we follow the below-given steps:

Step 1: Find the covariance matrix of the given feature matrix X.

Step 2: Then calculate the eigenvectors and eigenvalues of the covariance matrix. The eigenvector is the direction of best \mathbf{u} and the eigenvalue is the importance of that vector.

- The eigenvector associated with the largest eigenvalue indicates the direction in which the data has the most variance.
- Therefore, we can select our principal components in the direction of the eigenvectors having large eigenvalues and drop the principal components having relatively small eigenvalues.

2. t-SNE (t-Distributed Stochastic Neighbor Embedding)

- The core idea of t-SNE is to preserve the pairwise distance in a neighborhood when the datapoint in a higher dimensional space is projected into a lower dimensional space.
- Therefore, t-SNE tries to create an embedding that preserves the neighborhood using some probabilistic methods.

Ideas for the probabilistic model

- We need to define the probability that a point x_i is a neighbor of x_j .
- One way is to define probability as inversely proportional to the distance between points.

i.e.

$$p_{ij} = \frac{1}{\|x_i - x_j\|}$$

- But there are some problems with the above probability function:
 - The function gives an infinite value when the distance between the two points is zero.
 - Probability reduces too fast.
 - The function depends on an absolute function.
- Therefore, we take inspiration from Gaussian distributions for a better PDF and a new probability function is defined as:

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_i^2)}$$

- The standard deviation with respect to each x_i creates a scaled distance effect in the neighborhood of x_i and the denominator ensures that the sum of probabilities is 1.

Math for t-SNE

- Our objective is to project data points $x_i \in \mathbb{R}^d$ to y_i using t-SNE, where $y_i \in \mathbb{R}^{d'}$
- We compute the pairwise similarities as probabilities.
- We compute P_{ij} for d -dimensions and Q_{ij} for d' -dimensions where $d > d'$.

i.e.

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_i^2)}$$

where, p_{ij} is the probability that the points x_i and x_j are neighbors in d -dimensional space, and q_{ij} is the probability that the points x_i and x_j are neighbors in d' -dimensional space.

- We define q_{ij} with the same formulation as p_{ij} since every x_i and x_j would have corresponding y_i and y_j in d' dimensional space.

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_k - y_l\|^2)}$$

- Now, to get a useful d' transformation we need $p_{ij} \approx q_{ij}$. Thus, we need a loss function to optimize.

KL divergence

- It is a loss function that measures the dissimilarity between two distributions.
- The KL-Divergence between two distributions P and Q is written as:

$$KL - div(P_{ij}, Q_{ij}) = \sum_i \sum_j [P_{ij} \cdot \log(\frac{P_{ij}}{Q_{ij}})]$$

- Some interpretations using the KL-divergence formula are:
 - If P_{ij} and Q_{ij} are the same, then KL-divergence will be equal to 0.
 - If P_{ij} is very small and P_{ij} and Q_{ij} are different, then KL-divergence will have a small value.
 - Lastly, since KL-divergence is a measure of dissimilarity, it is always **greater than or equal to 0**.

- Here, P is working as a weightage because if P_{ij} is small we don't really care as points x_i and x_j will be far away from each other in d-dimension space.
- Thus, our **optimization problem** would be to find all the y_i s that **minimize** KL-divergence(P, Q).

T-distribution (Cauchy Distribution)

- Researchers proposed the use of t-distribution with a degree of freedom equal to 1 instead of gaussian distribution for dimension d' .

- New q_{ij} is defined as:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

- t-distributions work better than gaussian distributions with SNE because:
 - t-distributions with a degree of freedom equal to 1 have a longer tail than gaussian distributions
 - Gaussian distribution falls exponentially while t-distributions sort of inversely
 - So, for our q_{ij} s , if we use t-distribution, two points can go farther away and still get pairwise distance preserved of the sort. Therefore, it is able to capture distant neighbors better.
- Therefore, we use Gaussian distribution for p_{ij} s and t-distribution for q_{ij} s because of which if two points are far away in lower dimensional space, the probabilities will still remain the same.

Perplexity

- Perplexity can be interpreted as the effective number of neighbors whose distance we want to preserve
 - It is one of the parameters that we might want to configure when using t-SNE.
 - Its value is kept in the range [5, 50]
-

Recommender Systems

Recommender systems are systems that are designed to recommend things to the user based on many different factors. These systems predict the most likely product that the users are most likely to purchase and are of interest to.

Companies like Netflix, Amazon, Instagram, etc. use recommender systems to help their users to identify the correct product or movies for them.

Why recommender systems?

- Increase in revenue based on personalization.
- Better user experience.
- More time spent on the platform
- Help websites improve user engagement.

Recommender systems applications

- Netflix to recommend movies
- E-commerce websites to recommend the products.
- Social media platforms to recommend feeds/ blogs/news/songs...etc. Ex: Instagram, Facebook.
- Food recommendations by Zomato, Swiggy.

- Songs recommendations by Spotify and Wynk music.
- Dating apps recommending people.

Types of recommender systems

- 1) Apriori Algorithm
- 2) Content-based filtering system.
- 3) Collaborative-based filtering system.
- 4) Similarity-based filtering system.
- 5) Matrix factorization
- 6) Popularity-based recommender system.

1. Market-Basket Analysis

- Market basket analysis is used to analyze the combination of products which have been bought together.
- This is a technique used for purchases done by a customer. This identifies the pattern of frequent purchases of items by customers.
- This analysis can help to promote deals, offers, sale by the companies, and data mining techniques helps to achieve this analysis task.

Apriori Algorithm

- Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related

to one another. we can say that the apriori algorithm is an association rule learning that analyzes that people who bought product A also bought product B.

- It is a Frequency-based algorithm. Generally, the apriori algorithm operates on a database containing a huge number of transactions.
Ex: People who bought iPhones also bought Airpods.

Examples where the apriori algorithm can be used:

- Telecommunications
- Banking / Insurance
- Medical
- E-Commerce
- Retail

Association rules

- Association Rules are widely used to analyze retail basket or transaction data and are intended to identify strong rules discovered in transaction data using measures of patterns, based on the concept of strong rules.
- There are various metrics in place to help us understand the strength of the association between an antecedent and consequent:
- The **IF component** of an association rule is known as the antecedent. The **THEN component** is known as the consequent.

1. Support:

It is calculated to check how popular a given item is. It is measured by the proportion of transactions in which an item set appears. It is also used to measure abundance or frequency.

$$Support(x) = \frac{\text{Number of transactions with } x}{\text{Total number of transactions}}$$

Drawback: If 'x' is very popular then many items will have high support w.r.t to x.

2. Confidence:

It is calculated to check how likely item X is purchased when item Y is purchased. This is measured by the proportion of transactions with item X, in which item Y also appears.

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Number of transactions with } X \text{ and } Y}{\text{Number of transactions with } X}$$

Drawback: If 'y' is very popular then it will have high confidence w.r.t to many items.

3. Lift:

It is calculated to measure how likely item Y is purchased when item X is purchased while controlling for how popular item Y is.

$$\text{Lift}(X \rightarrow Y) = \frac{\text{Confidence}(X \rightarrow Y)}{\text{Support}(Y)} = \frac{\text{Support}(X \cap Y)}{\text{Support}(X) * \text{Support}(Y)}$$

- $\text{lift}(X \rightarrow Y) = 1$ if X and Y are independent
- $\text{lift}(X \rightarrow Y) < 1$, unlikely to be bought together.
- $\text{lift}(X \rightarrow Y) > 1$, likely to be bought together
- Gives Bi-directional recommendation.

4. Leverage or Piatetsky-Shapiro:

Leverage computes the difference between the observed frequency of X and Y appearing together and the frequency that would be expected if X and Y were independent. A leverage value of 0 indicates independence.

$$\text{Leverage}(X \rightarrow Y) = \text{Support}(X \cap Y) - \text{Support}(X) * \text{Support}(Y)$$

Though it is similar to lift, leverage is easier to interpret.

The leverage value lies in the range of -1 to +1, whereas the lift value ranges from 0 to infinity.

5. Conviction:

Conviction is another way of measuring association, although it is a bit harder to get your head around. It compares the probability that X appears without Y if they were independent of the actual frequency of the appearance of X without Y.

It can be calculated as the ratio of the expected frequency that X occurs without Y if X and Y were independent divided by the observed frequency of incorrect predictions.

A high conviction value means that the consequent is highly dependent on the antecedent.

$$Conviction(X \rightarrow Y) = \frac{1 - Support(Y)}{1 - Confidence(X \rightarrow Y)}$$

Steps to implement the apriori algorithm:

- 1) Create a frequency table of all the items that occur in all transactions.
- 2) Create a pivot matrix representing 1 if the item is present and 0 if the item is not present.
- 3) Encode the matrix, return 0 for all the items that have the value less than or equal to 0 and return 1 for all the values that are greater than or equal to 1.
- 4) Use the library mlxtend.frequent_patterns and import apriori.
- 5) Calculate frequent itemsets using the metric min_support.

$$min_support = \frac{\text{Number of times subset has occurred}}{\text{Number of transactions}}$$

- 6) Use the library mlxtend.frequent_patterns and import association_rules
- 7) With the help of association_rules select the appropriate metric (ex: support, confidence, lift ... default = confidence) to recommend the items.

Advantages:

- 1) It is the most simple and easy to understand and implement.
- 2) It doesn't require labeled data as it is completely unsupervised and hence this can be used for many different situations as we can find unlabeled data quite often.
- 3) It is used to calculate large item sets.

Disadvantages:

- 1) Apriori algorithm is an expensive method to find support since the calculation has to pass through the whole database.
- 2) It is computationally expensive.
- 3) Complexity grows exponentially.
- 4) Cold start problem.

2. Content-Based recommender System

- Content-based filtering makes recommendations based on the similarity of items. It uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.
- Content-based filtering does not require other users data during recommendation to one user.

Advantages:

- 1) The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.
- 2) No cold start problem.
- 3) No sparsity problem

- 4) The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.

Disadvantages:

- 1) Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge.
- 2) It always recommends items related to the same categories, and never recommend anything from other categories.

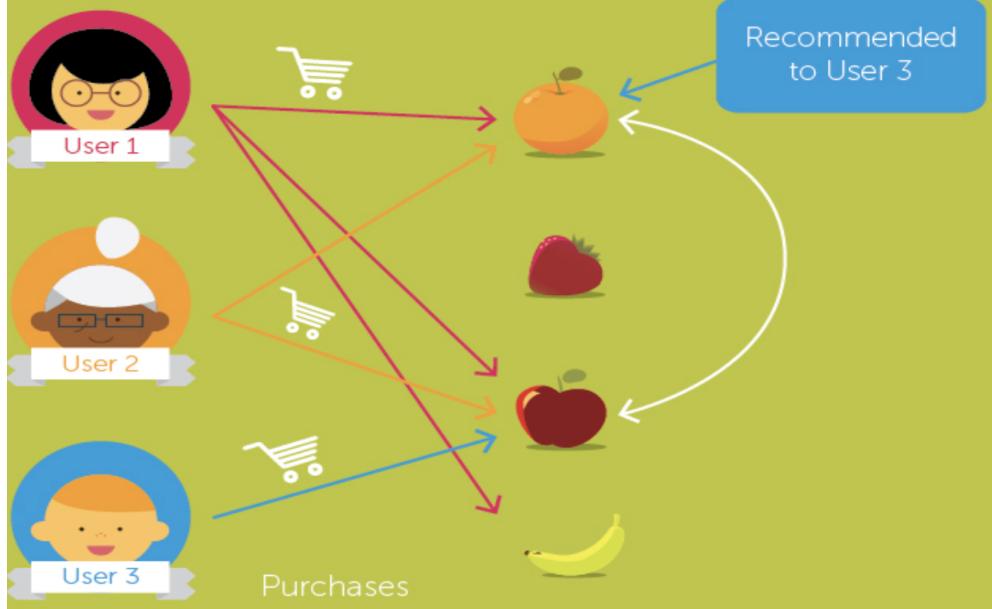
3. Collaborative Filtering System

- To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations.
- This system can filter out items that a user might like on the basis of reactions by similar users and makes recommendations based on user interactions.
- This filtering works on the assumption that people who like similar things have similar taste. There are different approaches we can adopt to implement CF recommender systems

3(a). Item - Item Based recommender System

- Item-item collaborative filtering, or item-based, or item-to-item, is a form of collaborative filtering for recommender systems based on the similarity between the items is calculated.
- “Users who liked this item also liked...”

Item-based filtering



Ex :

- User 1 bought Orange, Strawberry and Apple.
- User 2 bought Orange and Apple.
- User 3 purchases Apple so we calculate similarity between the fruits and recommend Orange.

To calculate similarity between two items, we use hamming distance.

- **Hamming Distance:** Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.

The Hamming distance between two strings, a and b is denoted as $d(a,b)$.

Ex: We have two strings in binary data

$$\text{Str1} = [0,0,1,0,0,1]$$

$\text{Str2} = [0,1,1,0,1,1]$.

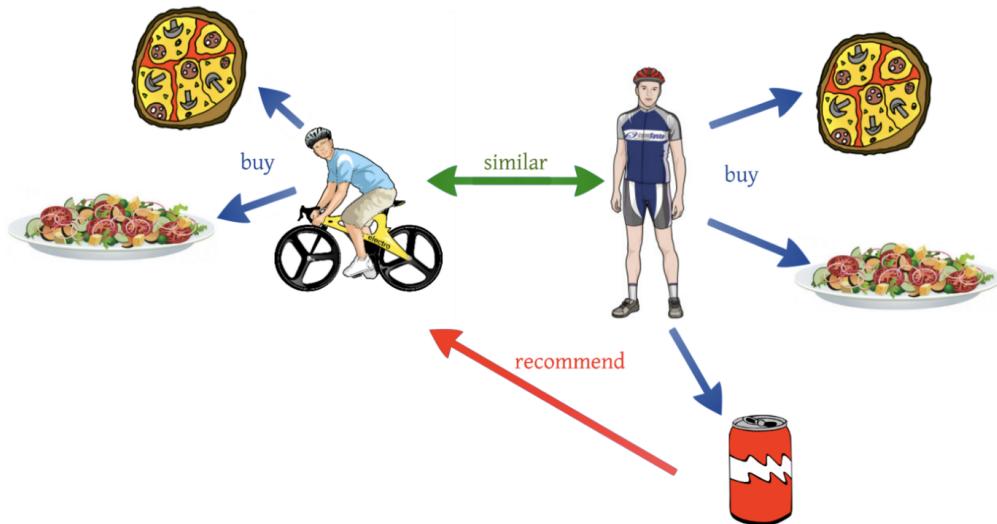
So, there are two positions in which bits are different hence the distance will be 2.

How Item-based works?

- Firstly, we compute similarities between items using hamming distance.
- Secondly, based on the computed similarities, items similar to already consumed/rated are looked at and recommended accordingly.

3(b). User - User Based recommender System

- User-user collaborative filtering, or user-based, or user-to-user, is a form of collaborative filtering for recommender systems based on the similarity between the users is calculated.
- “ Users who are similar to you also liked...”



Ex:

- There is user 1 who bought pizza, salad and soft drink and there is user 2 who bought pizza and salad. So, we calculate similarity between two users and recommend soft drink to user 2.

To calculate similarity between two items, we use euclidean distance or cosine-similarity.

- **Euclidean Distance:** The Euclidean distance is already familiar, the Euclidean distance is calculated between two 2-dimensional vectors $x = (x_1, x_2)$ and $y = (y_1, y_2)$ is given by:

$$\text{Euclidean dist} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- **Cosine Similarity:** Cosine similarity is the cosine of the angle between two vectors and it is used as a distance evaluation metric between two points in the plane.

Cosine similarity ranges from -1 to 1.

1 indicates the items are the same whereas -1 represents the compared items are dissimilar.

$$\text{Cosine similarity} = \frac{a \cdot b}{\|a\| \cdot \|b\|}$$

How User-based works?

- Firstly, we compute similarities between users using euclidean distance or cosine similarity.
- Secondly, based on the computed similarities between the users, the items of similar users are recommended.

Drawbacks:

- Data Sparsity: In case of a large number of items, the number of items a user has rated reduces to a tiny percentage making the correlation coefficient less reliable.
- User profiles change quickly and the entire system model has to be recomputed which is both time and computationally expensive.
- To overcome these drawbacks we use an item - item based recommender system.

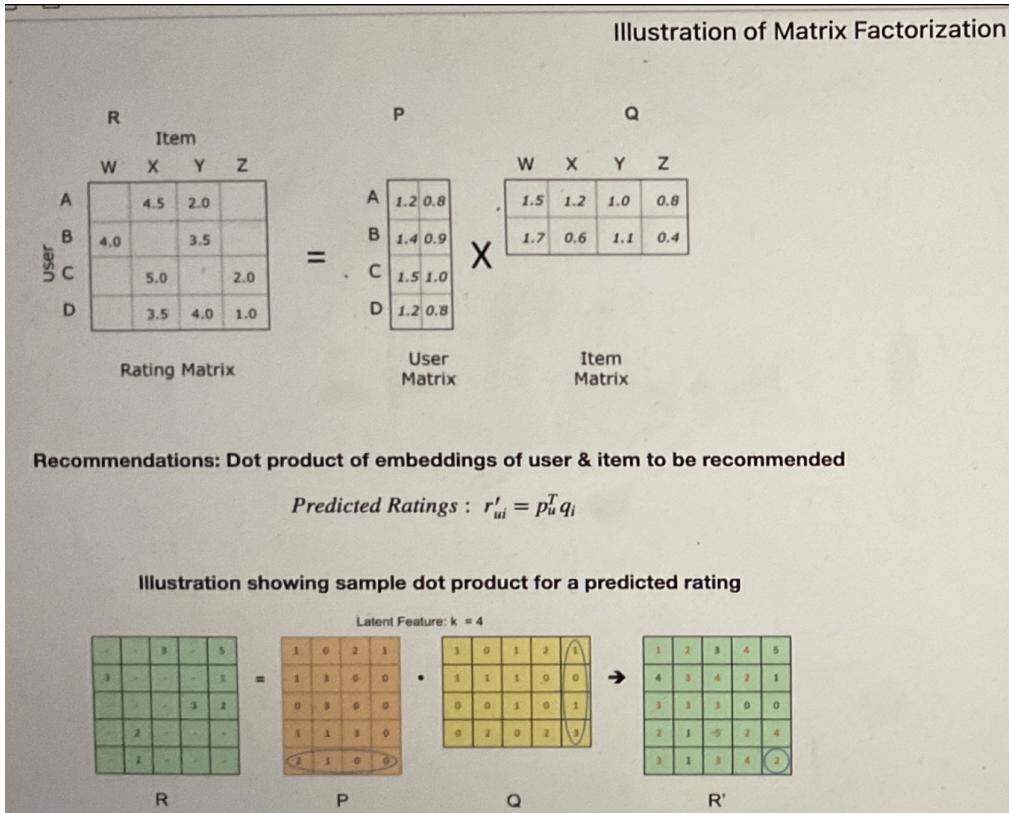
3(c). User - Item Based recommender System

- User-Item collaborative filtering is a form of collaborative filtering for recommender systems based on the similarity between the users and items is calculated. Data contains a set of users and items and ratings/reactions in the form of a user-item interaction matrix.
- If the user-item interaction matrix is sparse it is not easy to find the users that have rated the same items.
- To deal with this sparsity problem we use matrix factorization.

4. Matrix Factorization

- Matrix factorization is a way to generate latent features when multiplying two different kinds of entities. Collaborative filtering is the application of matrix factorization to identify the relationship between items and user entities.
- With the input of users' ratings on the items, we would like to predict how the users would rate the items so the users can get the recommender based on the prediction.
- Since not every user gives ratings to all the items, there are many missing values in the matrix and it results in a sparse matrix.

- Hence, the null values not given by the users would be filled with 0 such that the filled values are provided for the multiplication.



- Matrix P represents the association between a user and the features while matrix Q represents the association between an item and the features.
- We can get the prediction of a rating of an item by the calculation of the dot product of the two vectors.
- To get two entities of both P and Q, we need to initialize the two matrices and calculate the difference and we minimize the difference through the iterations. The method is called gradient descent, aiming at finding a local minimum of the difference.

$$\min_{U, I} \sum_i \sum_j (R_{ij} - u_i \cdot I_j)^2$$

- The user-item interaction matrix is very sparse since every user does not rate all items. The missing entries in the matrix would be replaced by the dot product of the factor matrices. Therefore, we know what to recommend to the users with the unseen movies based on the prediction.

Advantages of CF:

- 1) No domain knowledge necessary
 - We don't need domain knowledge because the embeddings are automatically learned.
- 2) Serendipity
 - The model can help users discover new interests. In isolation, the ML system may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item.
- 3) The system doesn't need contextual features.

Disadvantages of CF:

- 1) Fails in case of cold start: the model requires enough of other users already in the system to find a good match.
- 2) Collaborative filtering models are computationally expensive.
- 3) It's a bit difficult to recommend items to users with unique tastes.

5. How to evaluate the performance of recommender systems?

5(a). Qualitative evaluation through metrics

- There is a common practice to keep a hold out set and evaluate the quality using human evaluators
- Serendipity, Diversity, and Novelty are other attributes that are also important for a recommender system
- Serendipity: Ability of the model to help users discover new interests.

If the ML system treats the user in isolation, it may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item when it looks at aggregated preferences.

- Recommender systems that should suggest novel, relevant and unexpected items also help in diversification and popularity bias.

5(b). Quantitative evaluation through metrics

- **Precision at k:** Proportion of recommended items in the top-k set that are relevant

$\text{Precision}@k = (\# \text{ of recommended items } @k \text{ that are relevant}) / (\# \text{ of recommended items } @k)$

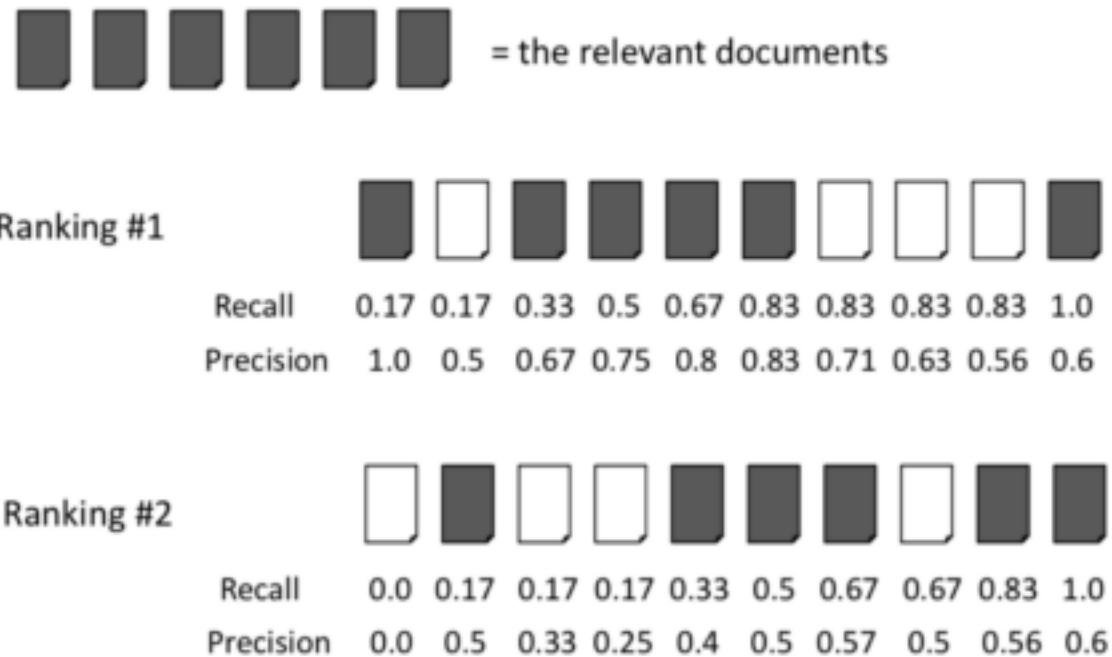
Ex: If precision at 10 in a top-10 recommender problem is 80% - 80% of the recommendations I make are relevant to the user.

- **Recall at k:** Proportion of relevant items found in the top-k recommendations

$\text{Recall}@k = (\# \text{ of recommended items } @k \text{ that are relevant}) / (\text{total } \# \text{ of relevant items})$

Ex: If recall at 10 is 40% in our top-10 recommender system. This means that 40% of the total number of relevant items appear in the top-k results.

- Illustration of precision and recall at 10 for two ranking models



- **Precision vs. Recall**

Precision as a metric in recommenders optimizes the ability of how good we are in retrieving relevant items (already well rated) but too much precision lowers the ability to give unique and new recommenders (false positives).

Optimizing for recall makes sense when the number of relevant items is less than recommended items. Here let us look at 100 recommend items & let us look at both precision and recall

- **Metrics for ranking:**

Coverage, Hit Rate, and F1 are some other accuracy-related metrics for recommender systems MAP, DCG, NDCG, and MRR are some other metrics that also considers order (rank of items) while evaluating recommender systems: Please refer to this article:

<https://towardsdatascience.com/ranking-evaluation-metrics-for-recommender-systems-263d0a66ef54>

Time series Analysis

Time series data

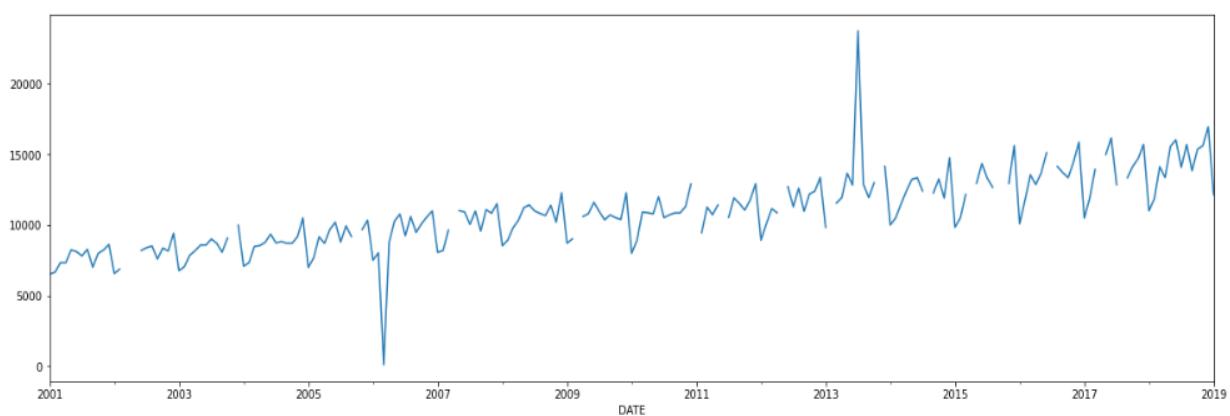
- A signal, indexed by an ordered timestamp is time-series data.
- For our data to be a time series, we need a minimum of two things
 - Date/timestamp (denoted as t)
 - One quantity (denoted as y)
- The timestamp(t) can be in days, weeks, months, years, or even seconds.
- Data observations like sales, revenues, inventories, etc are commonly expressed as a time series.

Forecasting

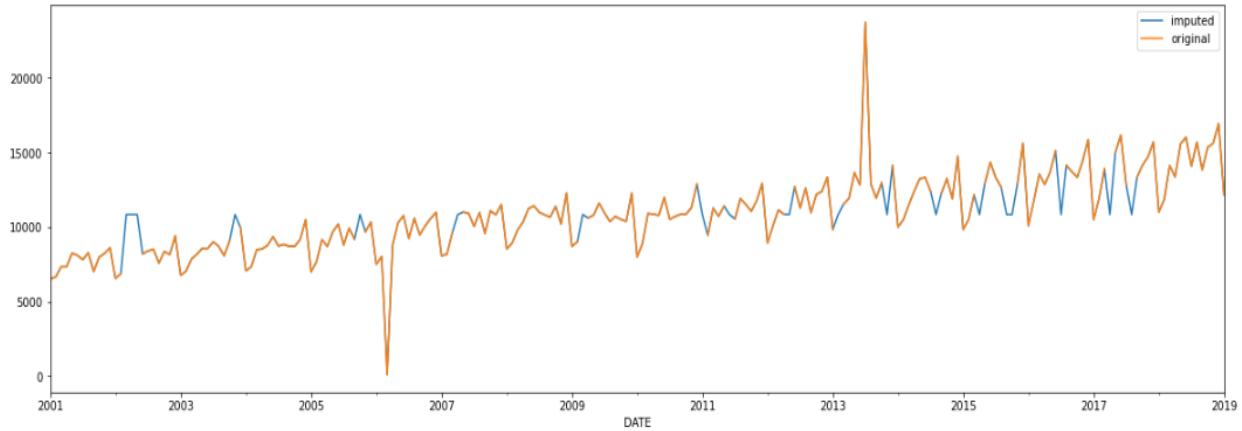
- Forecasting is analyzing historical data to predict future values. It is a supervised learning problem.
- It is more similar to regression in approach since we try to predict real-valued numbers.
- Given the historic values of a feature x till the present time t ($x_1, x_2, x_3 \dots x_t$), we need to predict the value of this feature for the future (x_{t+1}, x_{t+2}, \dots).
- In forecasting, Unlike regression, we do not have a set of inputs and one output. Rather, we have a signal and we are looking at some past values to predict some future value.

Handling missing values

Let's consider the sales dataset:



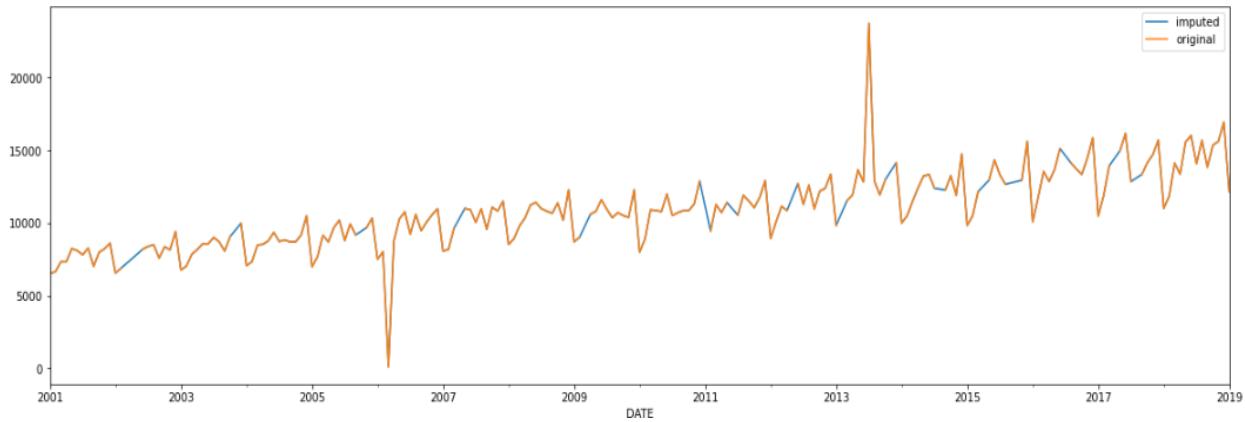
- One basic intuition is filling these missing values with the signal's mean or median. We will get the following plot after the imputation:



- We can observe that there are sharp increments and sharp decrements at some of the imputed places. This is because we imputed the local missing values with the mean of the entire data.
- Replacing with median would've also given the same result, as they have a similar value.

Linear Interpolation

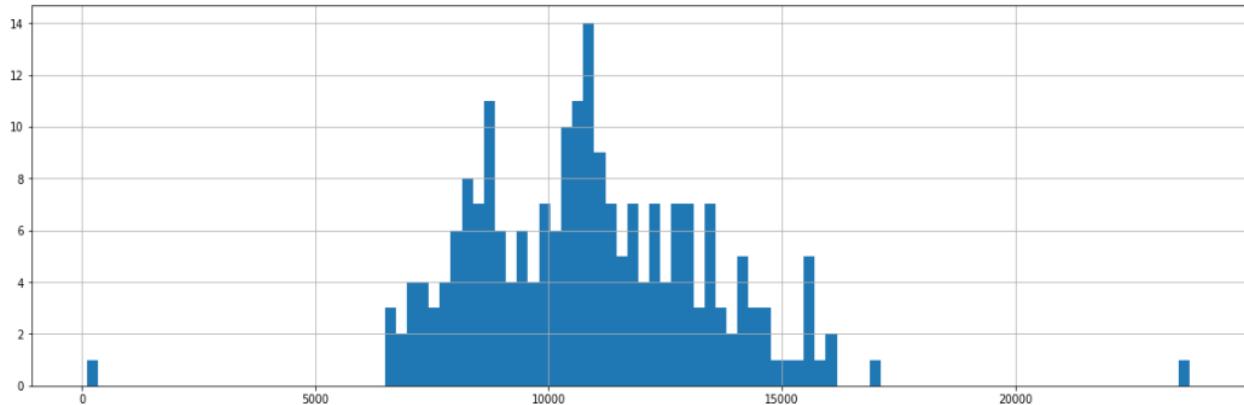
- It is a technique for handling the missing values. We take the average of the first point before and the first point after the missing value and fill the missing value with this average.
- It ensures that the missing values are not under or over-estimated by taking the average of the entire data.
- After using the linear interpolation for the imputation of missing values in the above data, we get the following plot:



- The imputed values fall within two finite points (preceding and succeeding points) because of which the imputed values don't seem to be forced.

Handling Anomalies

- Anomaly/outlier is an abnormal or unusual data point in the data set, which stands out of the data. It can simply be seen as a wrong entry in the data, and that can happen at times.
- Alternatively, it could be the correct entry but for a one-time event, which is not likely to repeat in the future. So, even if it's valid data, we would like to remove it from our training set, because we don't want our model to get biased by that one-time event that is not going to happen again.
- One of the best ways to identify the anomalies is by plotting a **Histogram**.

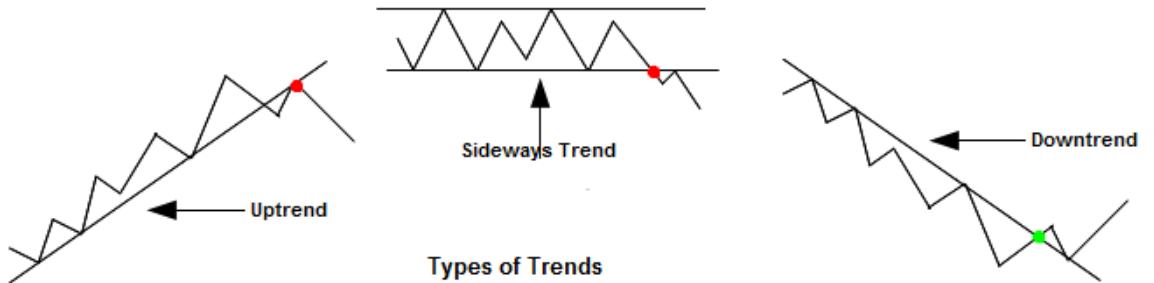


- From the plot, we can see the anomalies and we can easily decide where to cut the data, to get rid of anomalies.
- If the above histogram plot is more continuous, we use the concept of **quantiles** to deal with anomalies. We rule out observation as an anomaly if it is greater than 95 percentile, or less than 5 percentile.

Trend

- Trend can be thought of as the linear increasing or decreasing behavior of the series over a long period. The trend usually happens for some time and then disappears, it does not repeat.
- A trend can be uptrend, downtrend, or can be up and down, need not be a straight line.

- Trend line is a smooth predictable function that traces the trend of a time series, and can help us predict the time series indefinitely in the future.



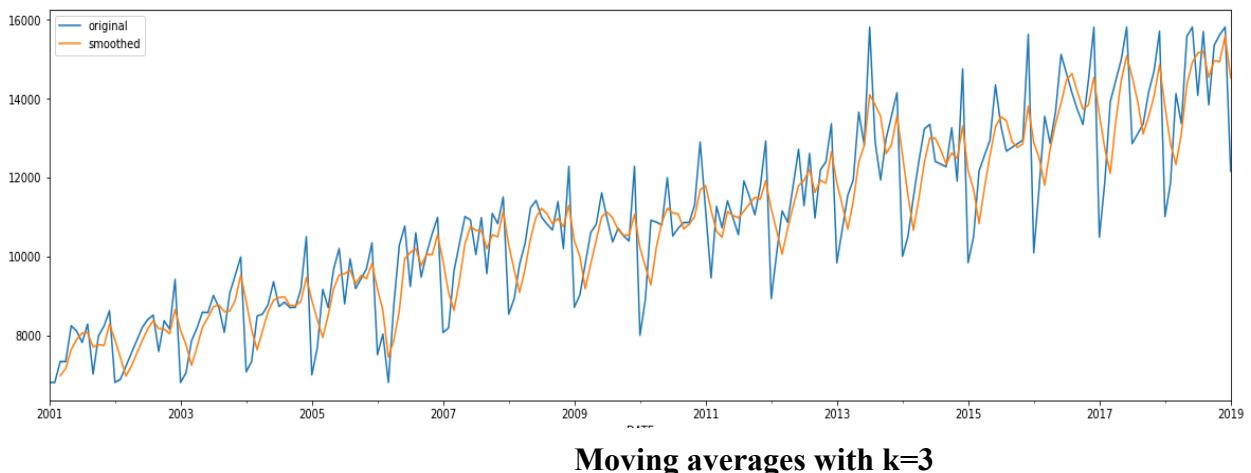
- Trend can be calculated by taking the rolling average (or moving average) over a long period or by just fitting a Linear Regression line on the points.

Moving average

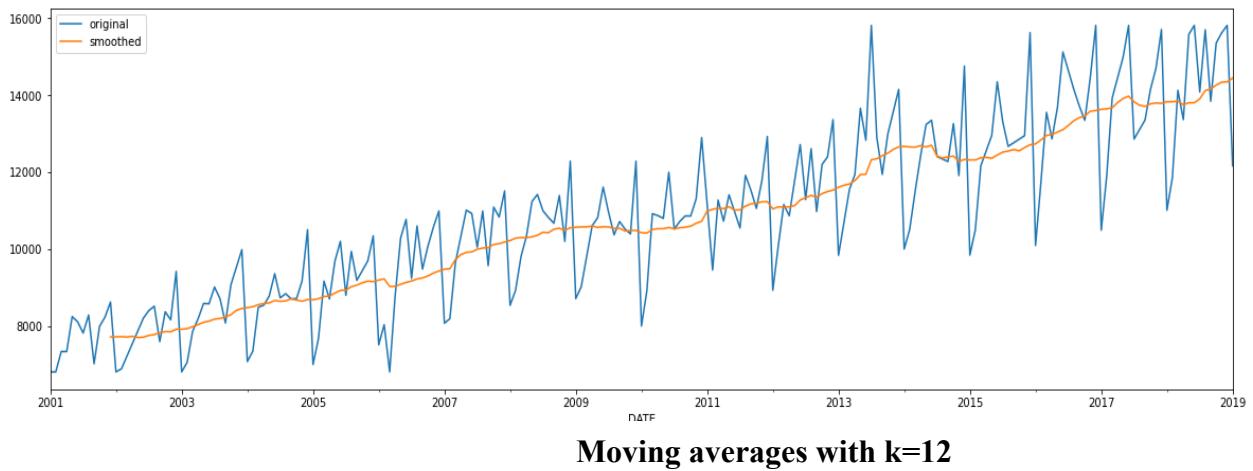
- If we take the average of the last k data points in our series and use it to guess the next point at $t=k$. This approach is called the Moving Average.
i.e.

$$y_t = \frac{1}{k} \sum_{i=1}^k y_{t-k}$$

- The value of k acts as a hyperparameter, which we can set based on what works best for us. It is also called the **window size**.
- The moving average line has been shown below.



- If we increase the window, the moving average curve gets smoother.



Centered moving average

- The average of n points before the current point, and n points after the current point is called the Centered moving average.
- We can't use this in forecasting as the future values are not available.

Seasonality

- Seasonality in time-series data refers to a pattern that occurs at a regular interval.
- Making copies of seasonality can fetch us our time series.
- A seasonal pattern occurs when a time series is affected by seasonal factors such as the time of the year or the day of the week.
- The steps for calculating the seasonality are:
 - Calculate the trend and subtract it from the original time series.
 - From the result, take the average across the period. If it is a monthly time series for 4 years, then you have 4 Jan, 4 Feb, 4 Dec, etc. Take the avg of all Jans, Febs, etc.

Time series decomposition

- There are three important components which we want to decompose our signal into.

- Trend - general movement over time
- Seasonality - behaviors captured in individual seasonal periods
- Residual - everything not captured by trend and seasonal components
- Time series decomposition is formulated in two different techniques combining these three different components:

Additive

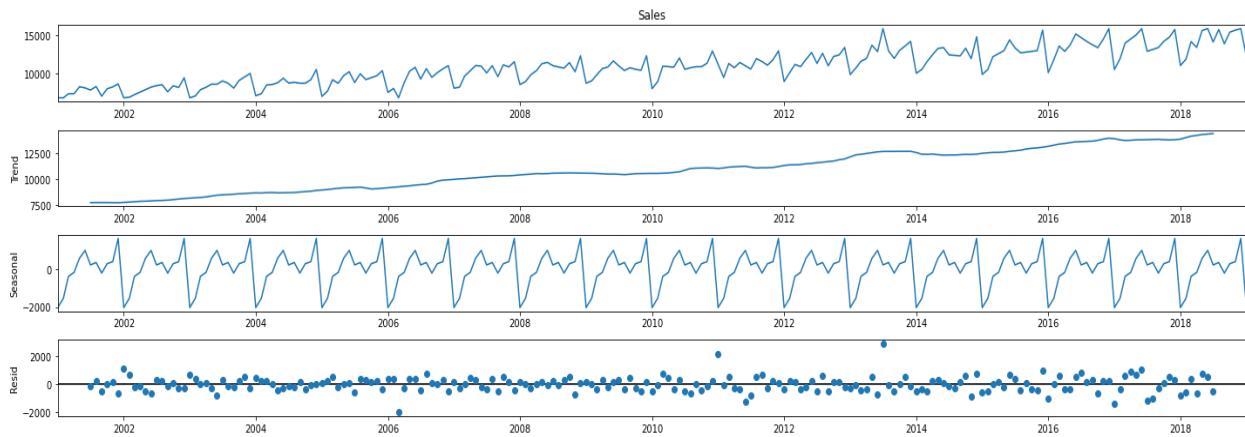
- $y(t) = b(t) + s(t) + e(t)$

Where, $b(t)$ -> trend of signal

$s(t)$ -> seasonality

$e(t)$ -> error term

- For the sales data, decomposition would be in the following way:



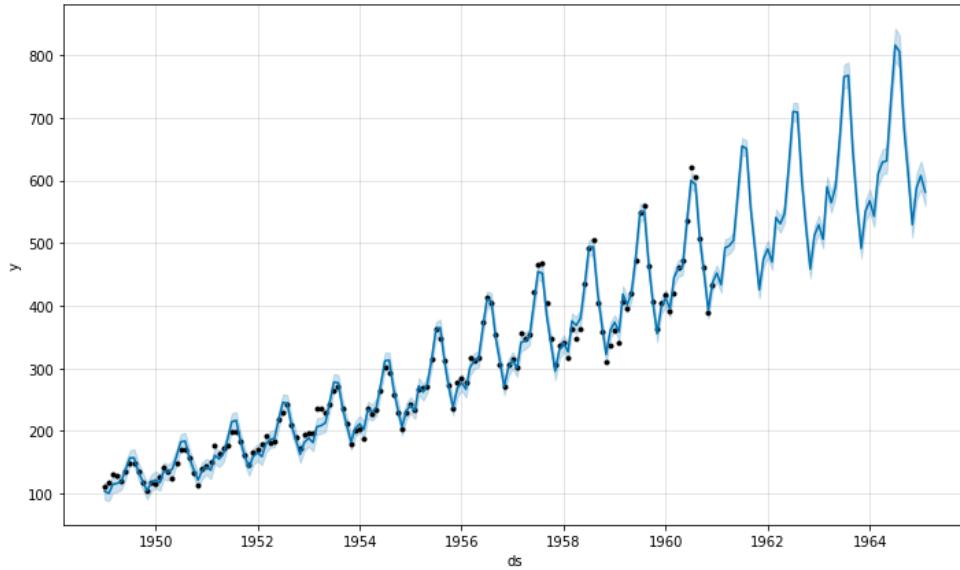
- The error term $e(t)$ cannot be estimated since it is based on the real values that we have but can be computed using the same formulation.

$$e(t) = y(t) - b(t) - s(t)$$

- If the trend and seasonality components obtained are a good estimate, then the errors would be small and would be scattered around zero.

Multiplicative

- It is a time series in which the amplitude of the seasonal component is increasing with an increasing trend.



Multiplicative seasonality Example

- In multiplicative seasonality, we do not obtain the time series by adding the trend and seasonality components with each other. Instead, if we multiply them, then we obtain a time series in which the amplitude of the seasonal component is increasing with an increasing trend.
- Therefore, the **multiplicative seasonality decomposition** is written as

$$y(t) = b(t) * s(t) * e(t)$$

$$e(t) = \frac{y(t)}{b(t)*s(t)}$$

Train test split

- For time series data, we do not perform a random shuffle for train and test data split.
- Random shuffling on time series data would be meaningless since it is a collection of data over time and we look at the past data to predict future values.

- Therefore, we do a **time-based splitting**. For example, for our sales dataset, out of the 18 years of data, we decide to train on 17 years of data and use the last year, i.e. the last 12 values for test data.

Sales	
DATE	
2018-02-01	11852.0
2018-03-01	14123.0
2018-04-01	13360.0
2018-05-01	15576.0
2018-06-01	15809.4
2018-07-01	14080.0
2018-08-01	15697.0
2018-09-01	13838.0
2018-10-01	15351.0
2018-11-01	15615.0
2018-12-01	15809.4
2019-01-01	12160.0

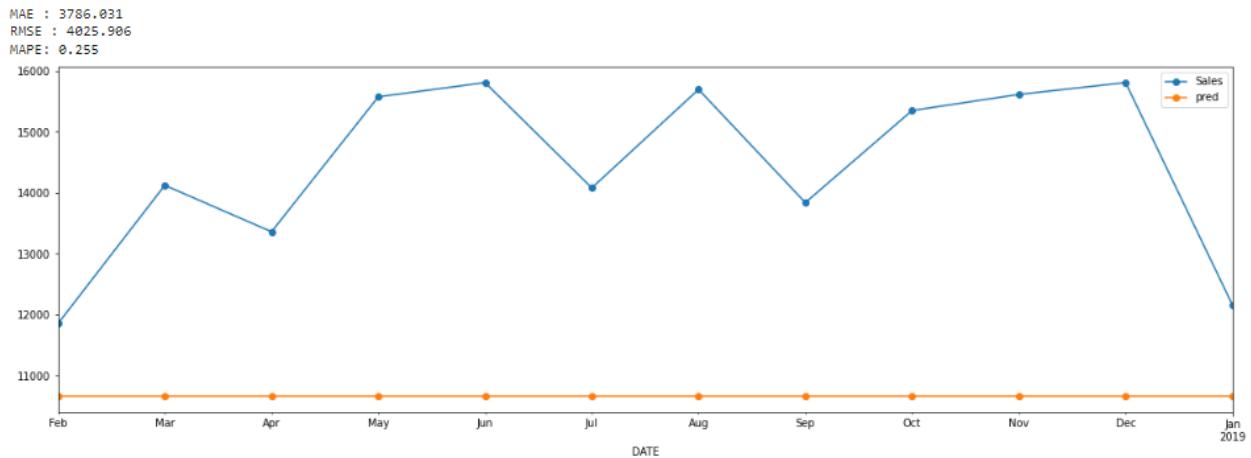
Test data

Simple Forecast Methods (not-so-intelligent approaches)

1. Mean forecast

- We take the average of all past k values of the time series to forecast the value at time $t=k+1$.

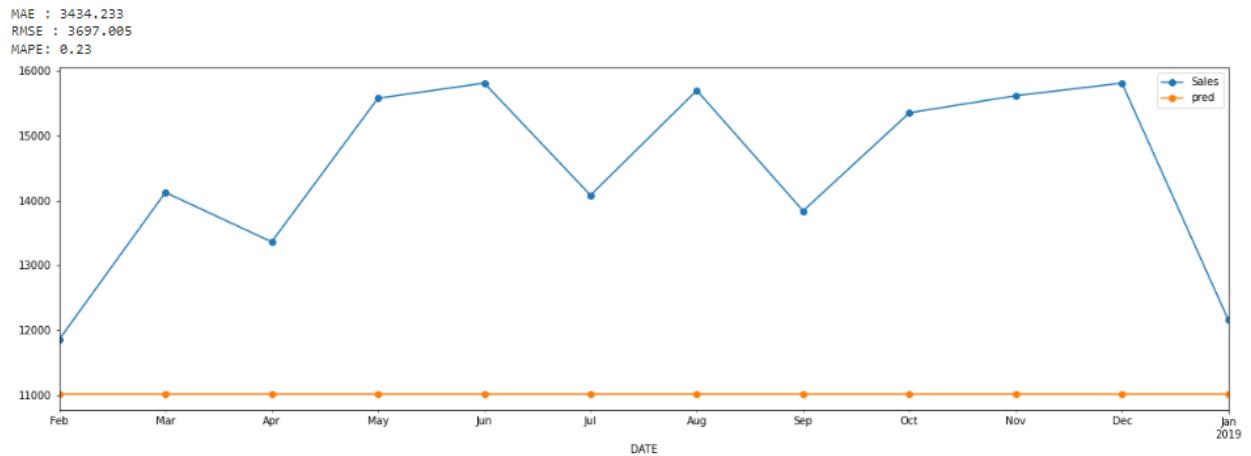
$$x_{k+1} = \frac{x_1 + x_2 + \dots + x_k}{k}$$



- The MAPE metric is showing that we have a 25.5% error, concerning its value. Thus, it is not a good model.
- A similar argument can be made for the median. So, we can rule out using the median of the entire data as future forecast values.

2. Naive Approach

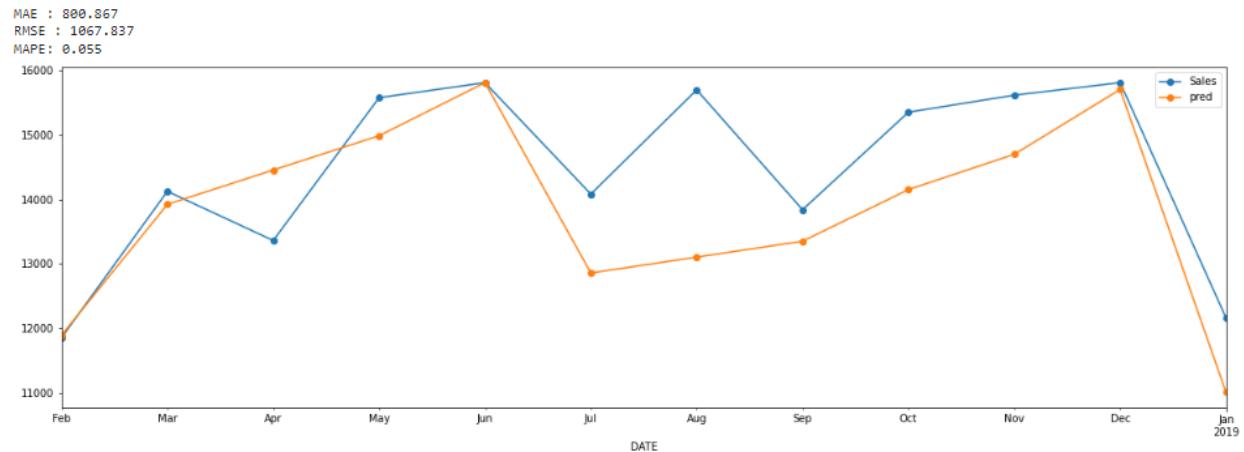
In this approach, we simply take the value of the series at time $t=k$ and forecast that for the future.



- This model's performance is slightly better(23%), but we can see it's still really bad.
- But if we get a different value for series at $t=k$, this would change all the future forecasts we had made earlier. Hence, this is not a very intelligent approach.

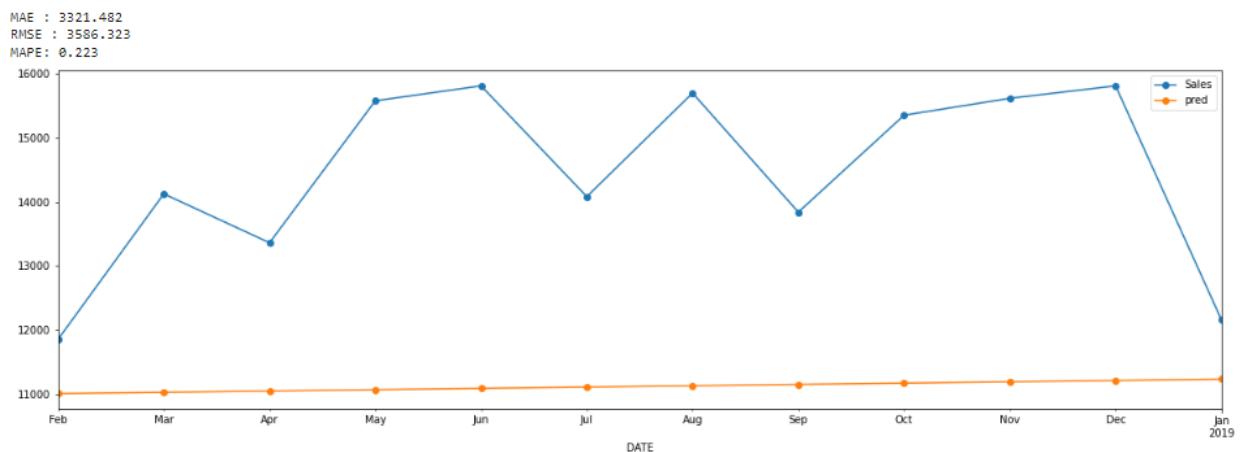
3. Seasonal Naive Forecast

- It is a smarter approach to optimize the Naive approach.
- We set each forecast to be equal to the last observed value from the same season (e.g., the same month of the previous year).
- This way, we essentially forecast the future values to be the same as last season.



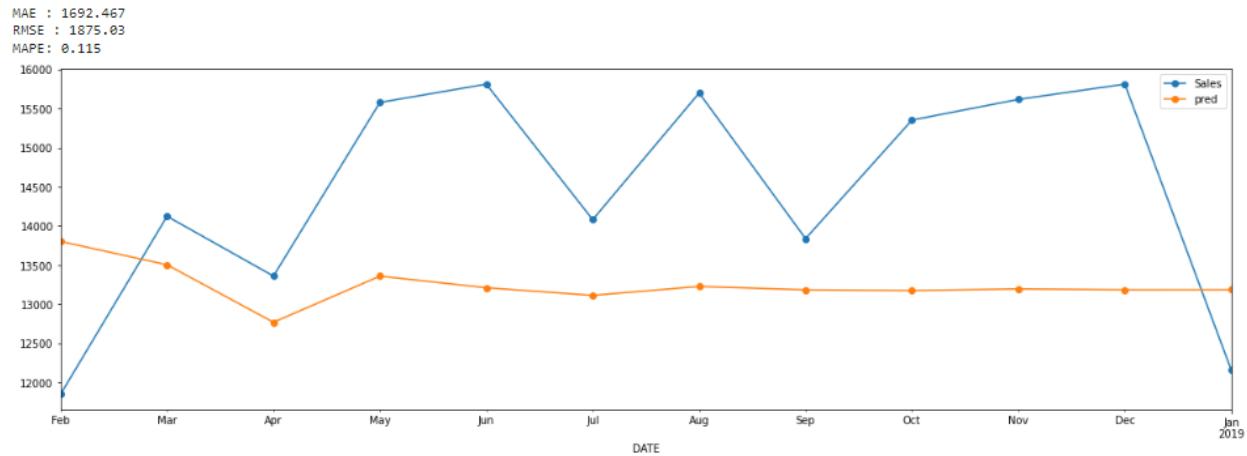
4. Drift Method

- In this method of forecasting, we take the first and last point of the data and draw a straight line between those points, and then extend this line into the future to get a forecast (**Linear extrapolation**).
- So, instead of just picking up some value from the past, we let our values increase or decrease over time.
- Depending on what the last point is, the drift (slope) may change significantly. thus, this is highly sensitive to the last value available.



5. Moving average

- The average of the last k data points in our series is used to forecast the next point in this approach.
- For our sales data, our model has an 11.5 % error when using this model. This is much better than the other simple models we saw.



- These were some simple models that do not give any smart forecasts.

Smoothing-based methods

1. Simple Exponential Smoothing (SES)

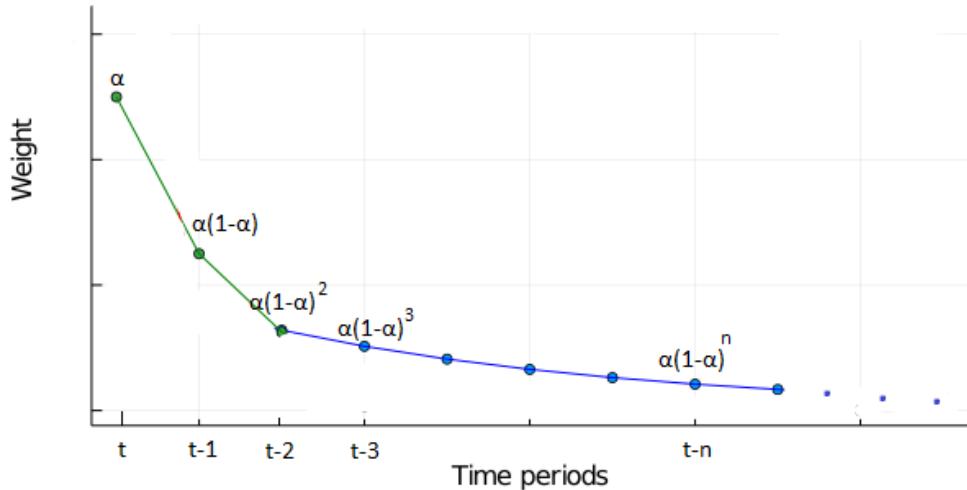
- The key idea of this method is to keep some memory of the entire time series, but also, we want to give more value to the recent data and less value to the past value. This forms a decaying trend.
 - Let's consider the weight we assign to the recent most value as α .
 α is called the **smoothing parameter**.
- So, our forecast at time t for the time $t+1$ is:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t$$

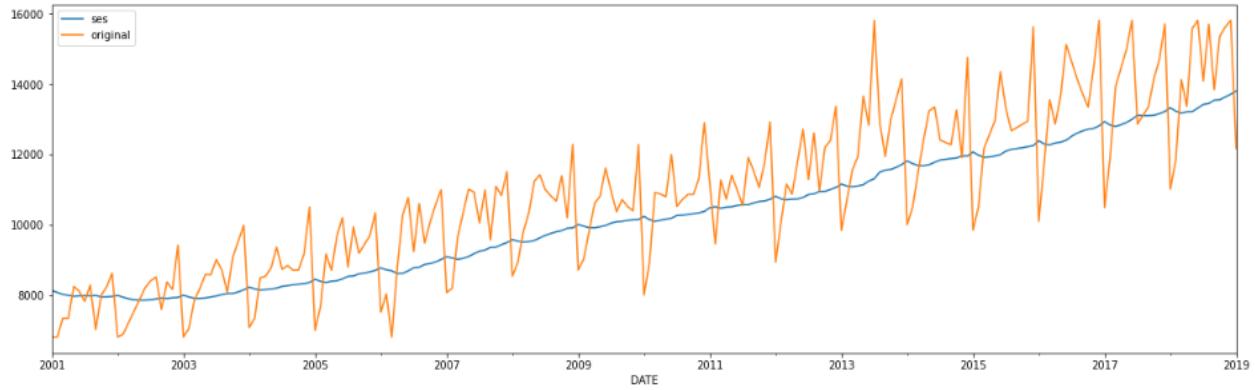
- The above formulation is recursive and expands in the following form:

$$\hat{y}_{t+1} = \alpha y_t + \alpha(1 - \alpha)y_{t-1} + \alpha(1 - \alpha)^2\hat{y}_{t-2} + \dots$$

- We can observe from the formulation that the weights are exponentially decaying. Therefore, we give more weightage to the most recent values, and this weightage keeps decreasing for earlier values.

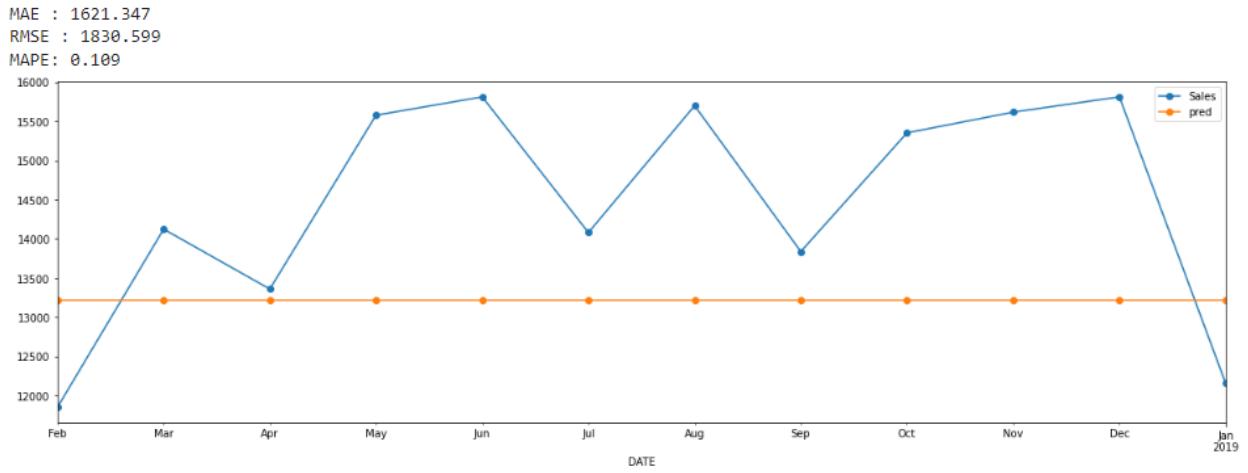


- Let's fit the SES model on the sales data:



Unlike the moving averages, it does not have the offset at the beginning and end, because this method is initialized properly.

- The forecast plot of this model on the sales test data is given below:



- The advantage of the above forecast is that the **level** of the forecasted values is right.
- However, the forecast is a completely straight line This is because we don't have the previous actual value available for horizon > 1. So the current forecast is used for all the next values.
- The prediction is a straight line, but the error is 10% which is less than the error of the moving average.
- Disadvantage of this model is that it is missing both trend and seasonality
- Now, we have the right levels, if we can predict the trend and seasonality right, we should get a good forecast.

2. Double Experimental Smoothing (DES)

- In this method, we incorporate the trend of the entire time series in the SES formulation to forecast future values.
- The weights are assigned to the trend value also and this forms an exponentially decaying series. Hence this is called Double Exponential smoothing (**aka Holt's method**). So basically, we are doing exponential smoothing on trend too.
- The formulation of DES is as follows:

$$\hat{y}_{t+h} = l_t + hb_t$$

where

$$l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$$

l_t is called the **level** of time series at time t.

- Calculation of the trend value:
 - Slope of a curve for $\Delta t=1$ is given as

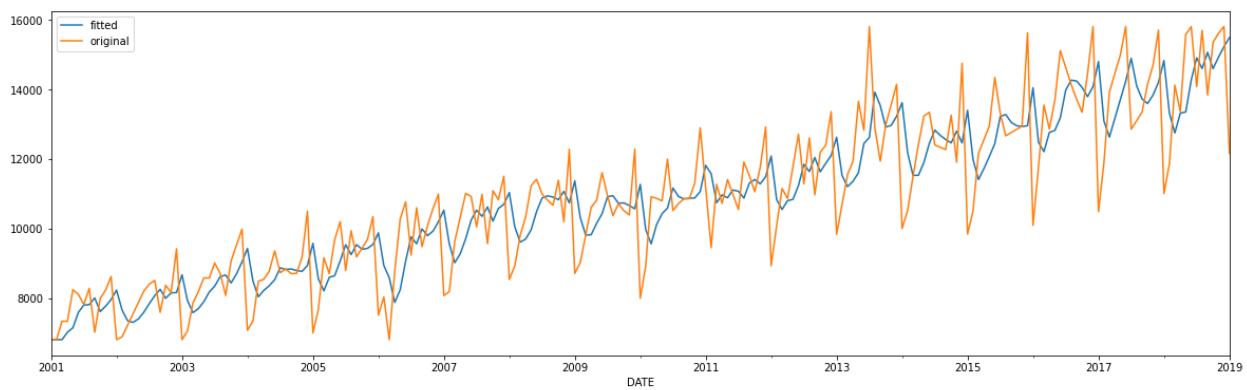
$$\frac{\Delta y}{\Delta t} = y_t - y_{t-1}$$

This slope value is equal to the trend of the series.
By plugging this in, we get,

$$b_t = \beta * (l_t - l_{t-1}) + (1 - \beta) * b_{t-1}$$

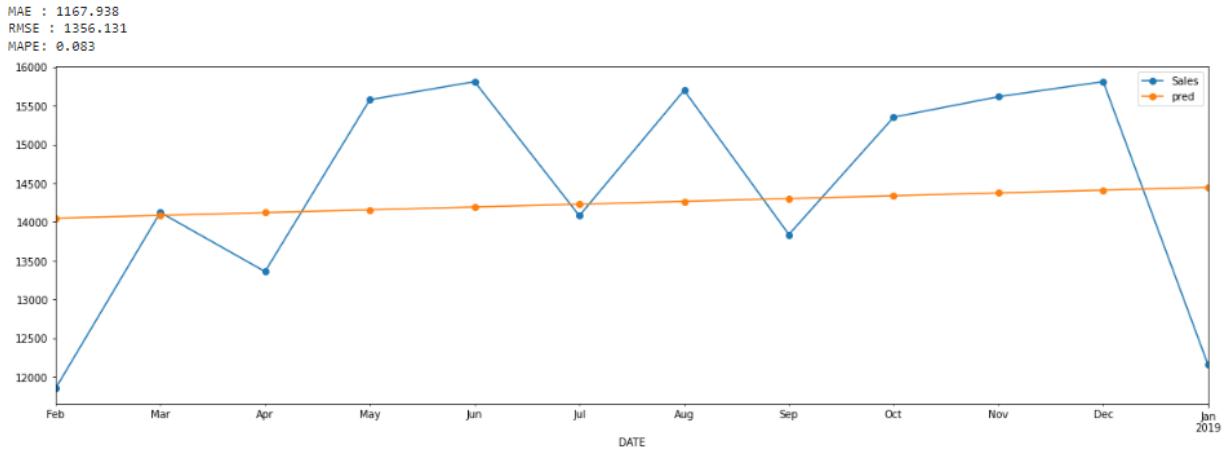
where, $(l_t - l_{t-1})$ is representing the current slope of the curve and b_{t-1} represents the previous slope.

- β is a parameter that needs to be tuned while training the model.
- Let's fit the DES model on sales data:



It is a better fit than Simple exponential smoothing.

- The performance of the DES model on the test set:



- The error (8.3%) is less as compared to the SES model (10%).
- Disadvantage of this model is that it does not consider the seasonality of time series.

3. Triple Exponential smoothing (aka Holt-Winters Method)

- Triple Exponential Smoothing is an extension of Double Exponential Smoothing that explicitly adds support for **seasonality** to the univariate time series. The seasonality value of the entire time series is also incorporated in this model.
- Upon incorporating the seasonality, our equation becomes,

$$\hat{y}_{t+h} = l_t + h b_t + s_{t+h-m}$$

where, **m** -> frequency of the seasonality

Therefore, s_{t+h-m} is representing the smoothed seasonality.

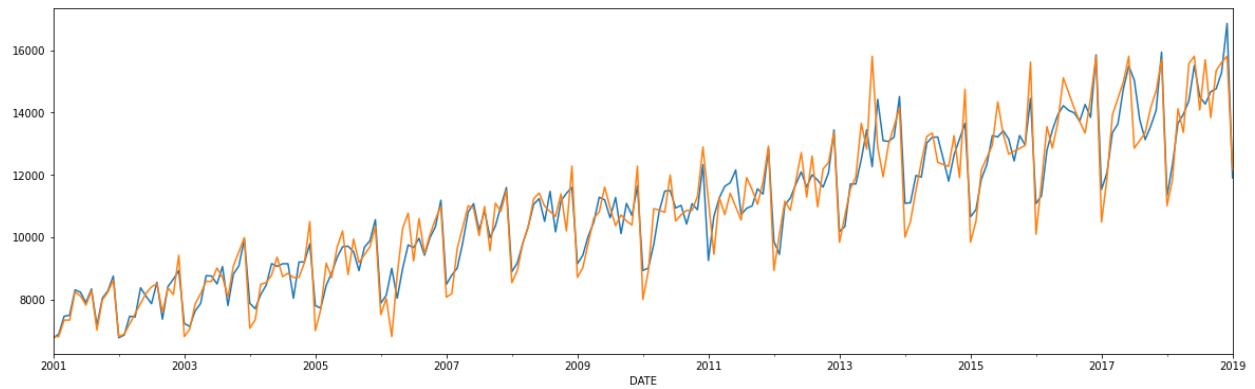
Also,

$$l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + b_{t-1})$$

$$b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_{t-1}$$

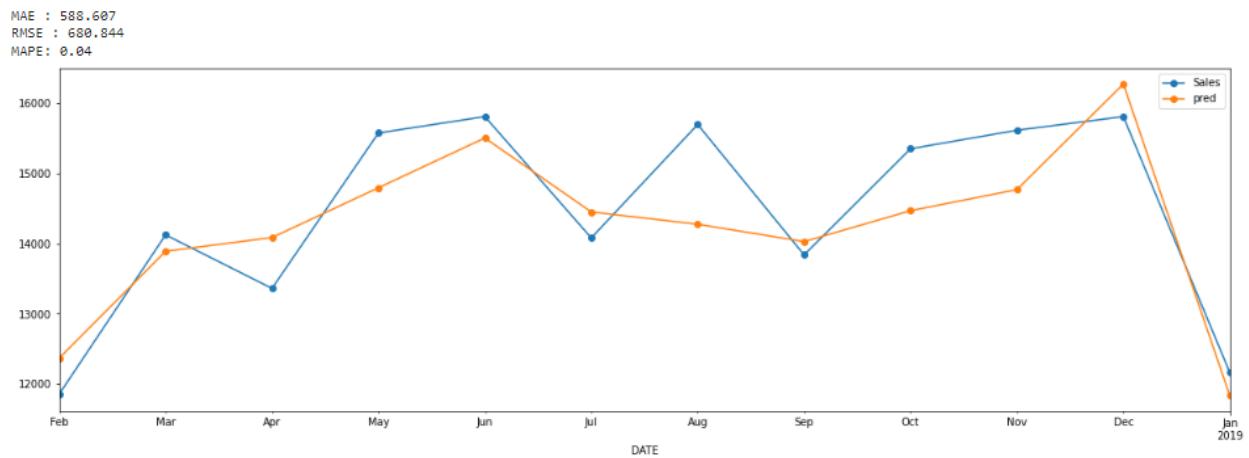
$$s_t = \gamma(y_t - l_t - b_t) + (1 - \gamma)s_{t-m}$$

- Let's fit the TES model on sales train data:



It can be clearly observed that this model captures more information than Double Exponential smoothing.

- Let's look at the forecasts of the TES model on the sales test data:



We can clearly see that this model has a better performance as compared to SES and DES.

The MAPE error is only 4% now.

- We can take a mixture of additive and multiplicative models. There is no rule for which model (multiplicative/additive) to use when. We need to try and see which performs better.

Stationarity

- For a time-series model to be stationary, the parameters (like mean, variance, amplitude, and frequency) of the models should **not be dependent on time**.
- Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times.
- **For example** Heartbeats (mean=0; standard deviation=1) are stationary — it does not matter when you observe it, it should look much the same at any point in time.
- In general, a stationary time series will have no predictable patterns in the long term.
- If a model's parameter varies over time, then there is a complex relationship that needs to be modeled, which all models are not able to account for.
- Many models assume the series to be stationary to be able to give useful results.
- So, either we want to have a stationary time series, or convert to it. Categorizing a time series by just looking at it can be a little subjective.

Dickey-Fuller Test

- There is a **statistical** method called the **Dickey-Fuller test**, which is designed for testing for stationarity.
- It fits an auto-regressive model and checks if it worked or not. If it did, then that means it was a stationary time series.
- There is a complicated mechanism to it. We don't need to know how it works. Just need to be aware of this test, as it can be handy.
- **How to implement Dickey-Fuller Test?**
 - We can find this as a built-in function under the **statsmodels** library as `sm.tsa.stattools.adfuller()`.
- **How do we interpret the result of the Dickey-Fuller test?**
 - This test returns the **p-value**. For a time series to be stationary, the **p-value** should be less than 0.05

Converting a Non-stationary Time Series to a Stationary Time Series

- As per basic intuition, if we remove trend and seasonality from our time series, it should become stationary.
- If it's still not stationary, then maybe there's still some seasonality or trend component left in the series.
- Ideally, the trend gets removed in one step. However, seasonality can take multiple steps to be removed.
- These processes are called **Detrending** and **Deseasonalising** respectively.

De-Trending

- The trend-seasonality decomposition that a time series can be written as:

$$\hat{y}_t = m * x_t + c + s(t) + r(t)$$

Where; $m * x_t + c$ represents the trend component.

- To remove this, we can **differentiate** the above equation concerning time t. This way, we'll get a stationary time series. This gives us:

$$\frac{d\hat{y}_t}{dt} = m,$$

$$\frac{d\hat{y}_t}{dt} \approx \lim_{\Delta t \rightarrow 0} \frac{y(t+\Delta t) - y(t)}{\Delta t}$$

- The minimum value of Δt that we have is 1, since we're talking in terms of time, and our minimum step is 1 month. So the equation becomes:

$$\frac{d\hat{y}_t}{dt} \approx \frac{y(t+1) - y(t)}{1}$$

- This process is called **differencing**.
- **NOTE:**
 - Differencing gives a good approximation of differentiating

- Differentiating gives us a Detrended time series.

$$y(t) = b(t) + s(t) + e(t)$$

\downarrow
 straight line
 $\approx y = mx + c$

differentiate

$$y'(t) = \text{const} + s'(t) + e'(t)$$

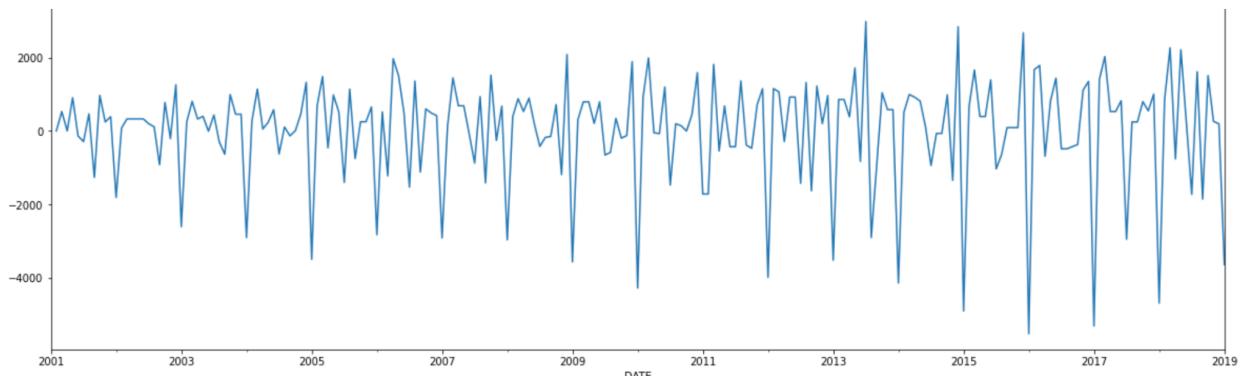
Trend is gone !!

- If the time series has a non-linear trend, then we'll have to differentiate it multiple times, in order to finally achieve a stationary series.
- NOTE:**
 - If the trend is an exponential function, then we'll not be able to convert it into stationary.
 - This is a very rare case.
 - Some exponential functions can be approximated by polynomials. In that case, we differentiate this polynomial to obtain a stationary time series.
- For applying differencing, a new series is constructed where the value at the current time step is calculated as the difference between the original observation and the observation at the previous time step.

$$\text{value}(t) = \text{observation}(t) - \text{observation}(t-1)$$

$$y'[t] = y[t] - y[t-1]$$

- For this, We use the **diff()** method of **pandas**.
- This method calculates the difference between a Dataframe element compared with another element in the Dataframe (default is an element in the previous row, as the default value is 1).

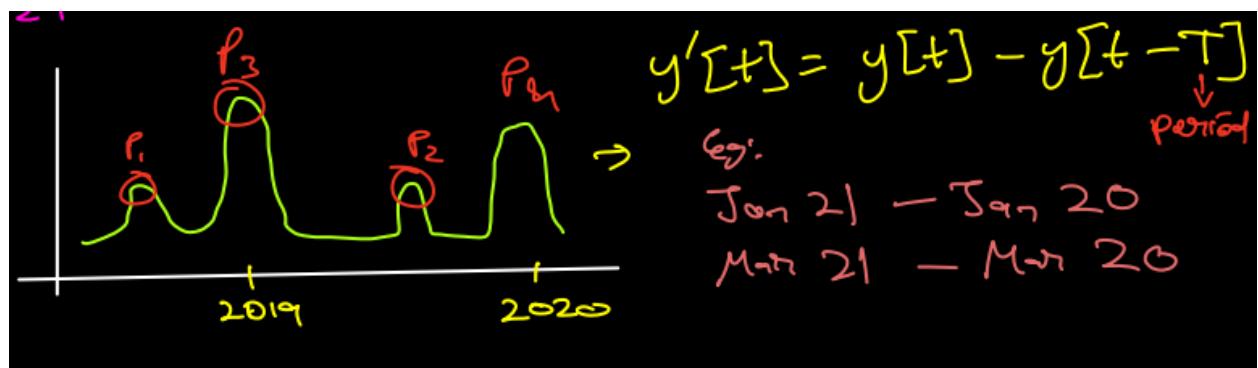


De-Seasonalising

- We can use differencing, but instead of subtracting from the last point, we need to take a difference from the last mth point, where m is the period of seasonality of the series.

$$\Delta y = y_t - y_{t-m}$$

- This is called **m-differencing**.
- If there is a seasonal component at the level of one week, then we can remove it on an observation today by subtracting the value from last week.

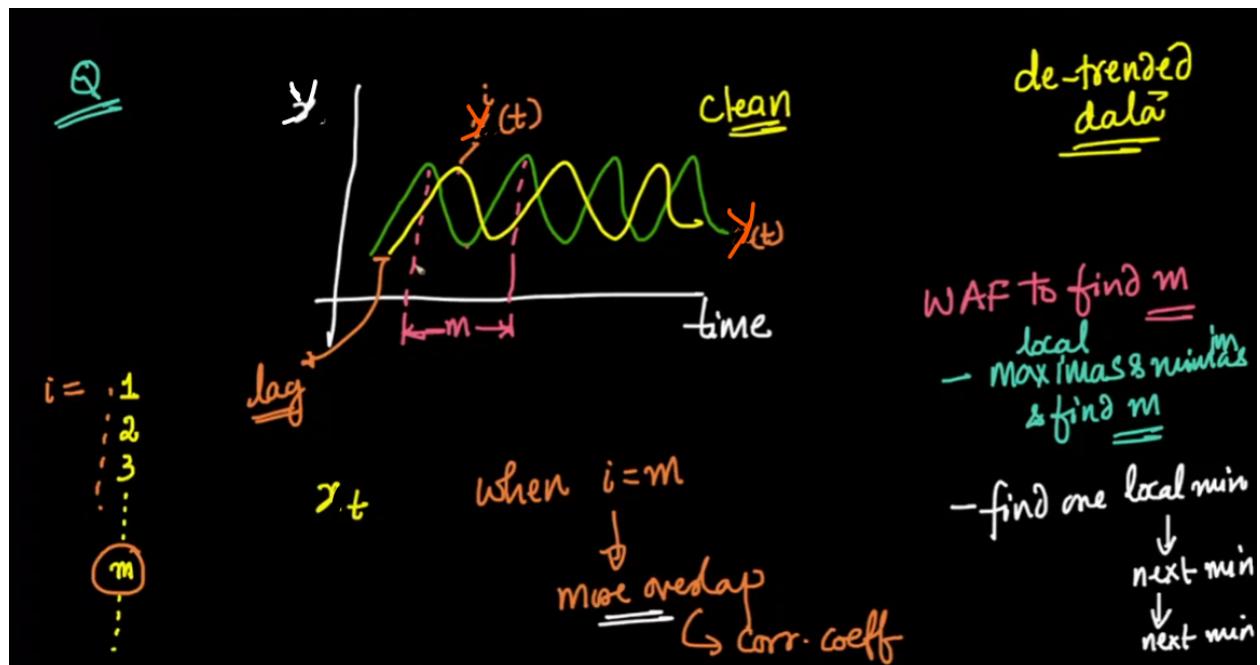


Autocorrelation

- One approach to find the value m is that we find local maxima and minima in the time series, and try to analyze the intervals at which they are observed.
 - This approach makes sense.
- Another approach would be; given a time series $y(t)$,
 - What if we consider another time series where we introduce a **lag** of 1, i.e. **shift the series** by 1 unit of time: $y^1(t)$
 - We can then find the **correlation coefficient** between these two time series: $y(t)$ and $y^1(t)$.
 - Similarly, we find the correlation between the original time series $y(t)$ and a time series lagged by i units: $y^i(t)$, where;

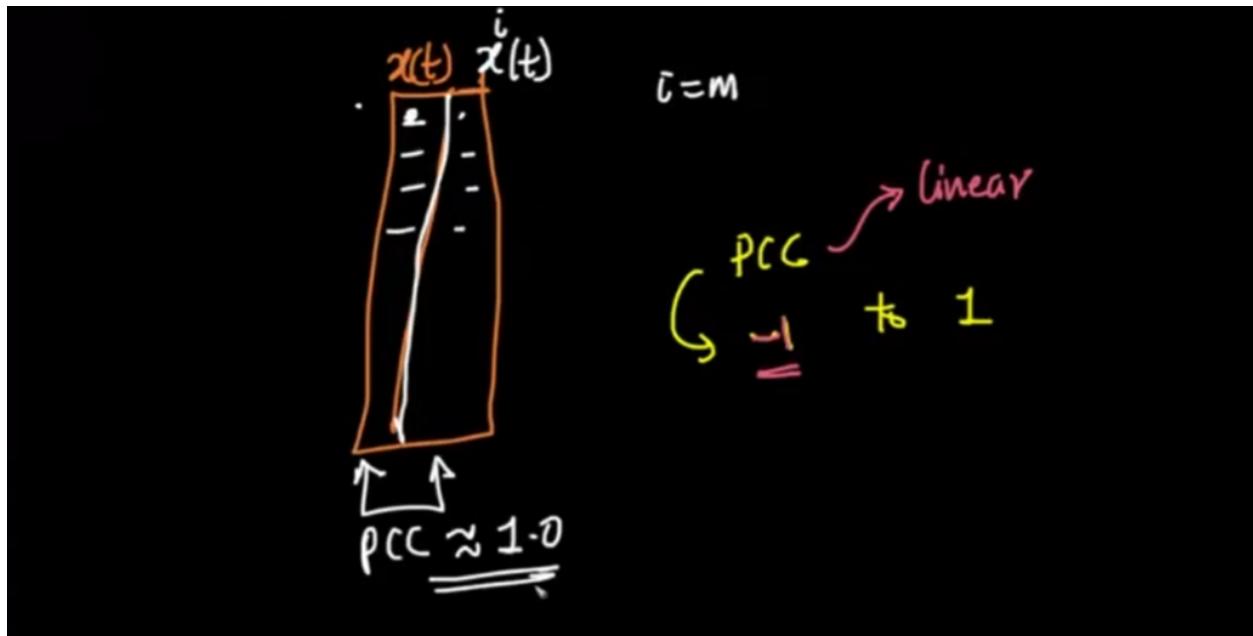
$i=1,2,3,\dots$ (i represents lag)

- In doing so, we would find a value of i , where the lagged time series **roughly overlaps** over the original series.



- Then, we create a table containing: $y(t)$ and $y^i(t)$ time series values
- When $i=m$, (the lagged time series roughly overlaps over the original series), the Pearson correlation coefficient would be very close to 1.

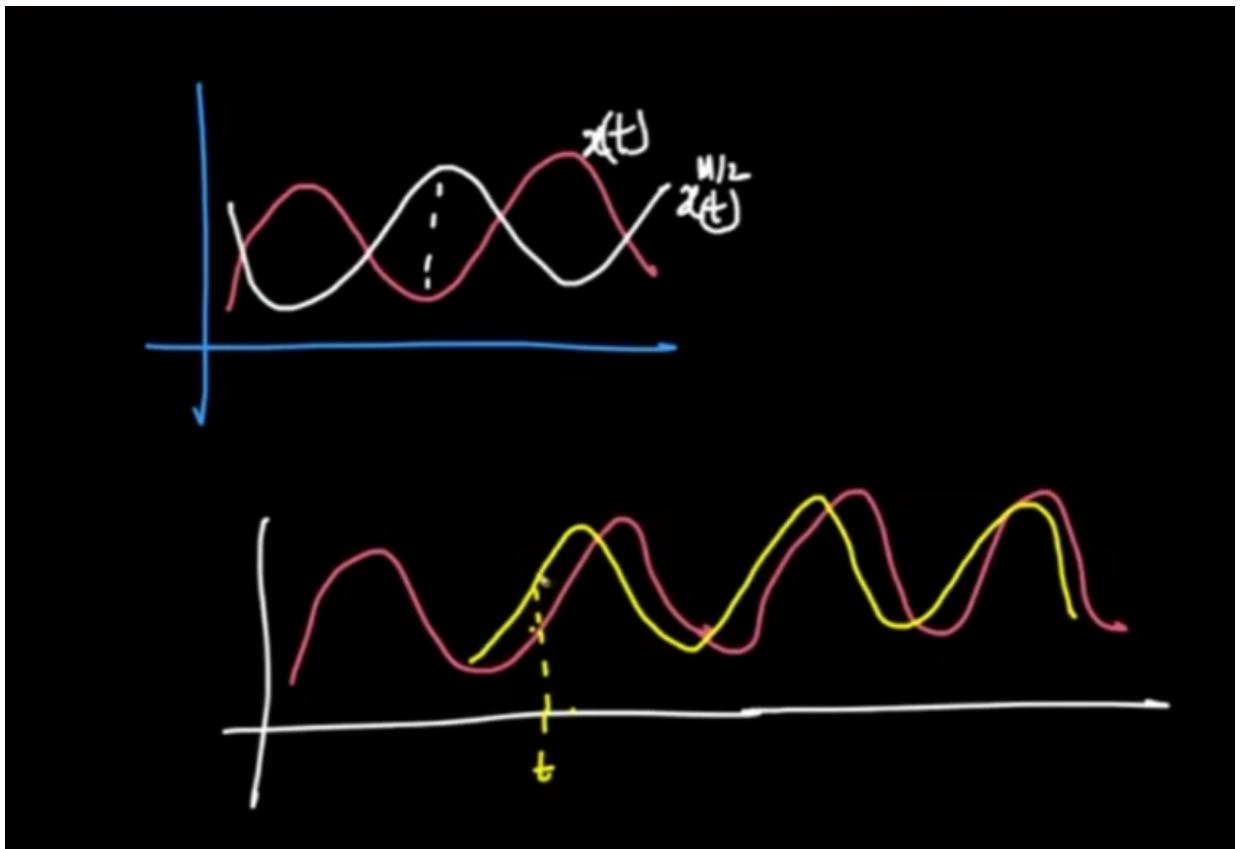
- This value of i would indicate the optimal value of m .



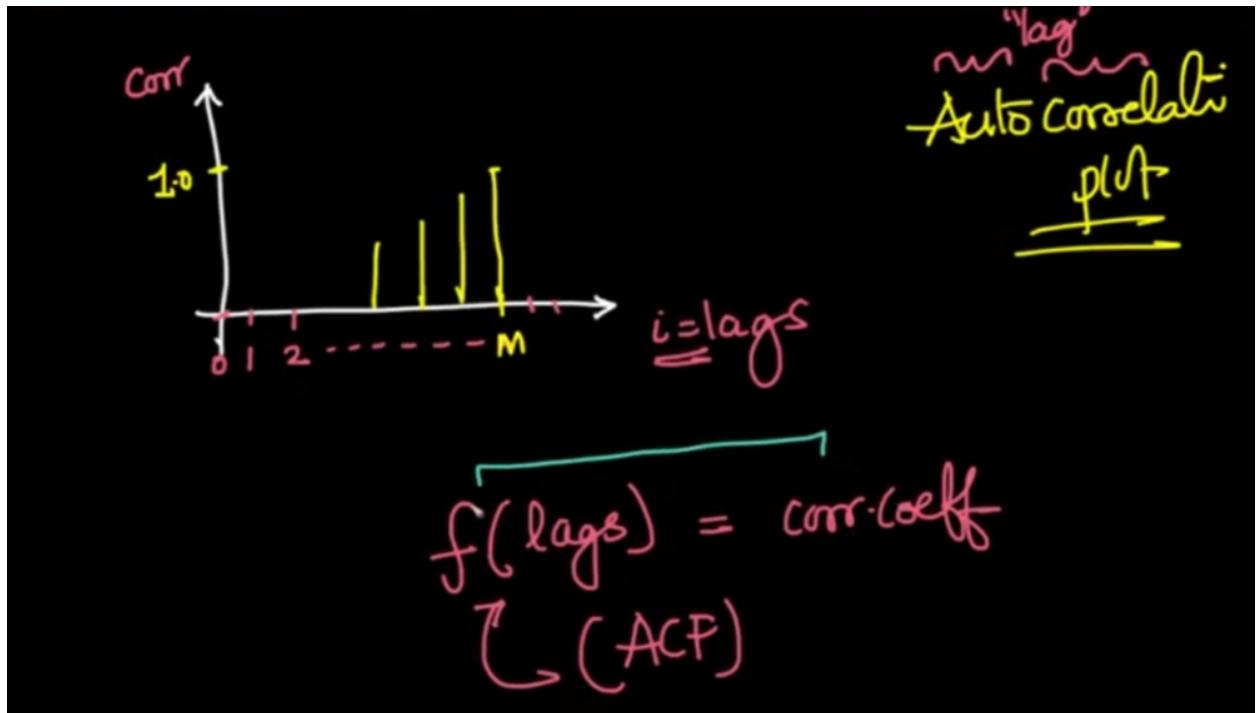
- What are the advantages of using the concept of correlation for finding the optimal value of m ?
 - Easily interpretable
 - Value ranges from -1 to +1
 - Captures linear relationship

Q. What will the plot between the correlation coefficient and lag (i) look like?

- At $i=m$, we would get a correlation value very close to 1.
- At $i=m/2$, as the value of lagged time series increases, the value of the original series decreases, giving us a **strong correlation**.
- For a value of i that is even close to m , though the correlation value would not be as strong as at m , it would be relatively strong.

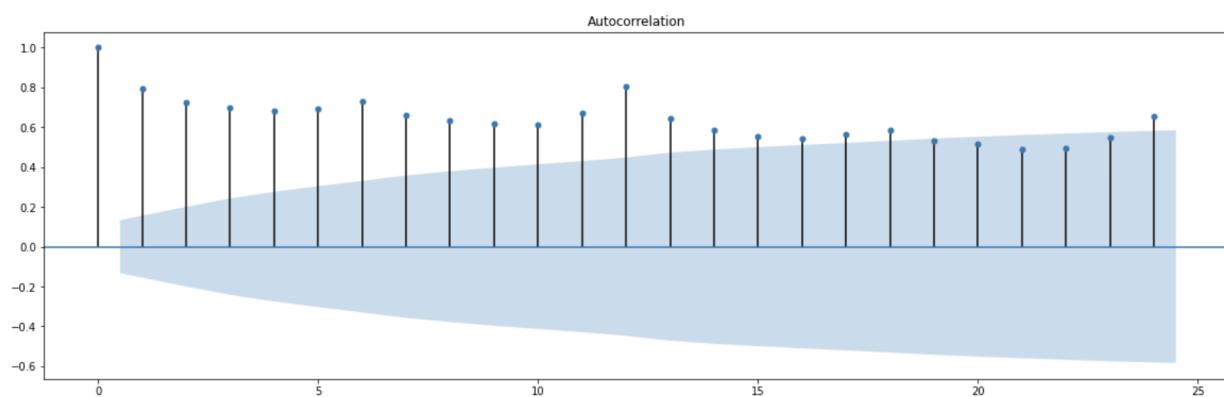


- So, the final plot between lag value (i) and correlation coefficient would look something like this:-

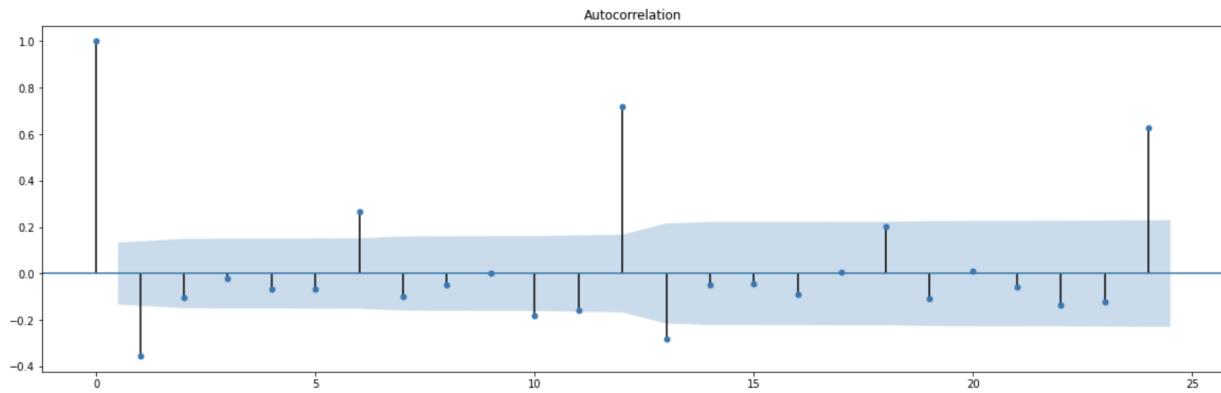


- This plot is called an **Autocorrelation Plot** because we are computing the correlation of the series with itself with various values of lag.
- This can be written as a function also. That is called **Autocorrelation Function (ACF)**.

ACF Plot of Trended Series:



ACF Plot of De-Trended Series:



- After detrending the series, we can see the correlation of the series is high at the period of 12 and 6. This indicates there is some seasonality.
- Besides that, the correlation values are random and small.
- On removing the trend, we're **able to capture negative correlations** between the original series and lagged series with more ease.
- The blue color highlighted part is the **confidence interval** which gives the significance level of the correlation
- If the correlation is higher or outside of the blue highlighted area, that can be considered as a highly significant value.
- So the major values of lag we consider to study the seasonality pattern here are:-
 - Correlation at lag=1
 - Correlation at lag=6
 - Correlation at lag=12
 - Correlation at lag=24
- Other points lie within or very close to the blue shaded region of confidence intervals.
- This means that we are not very confident about these correlation values
- The confidence interval increases as we consider time series that lagged more

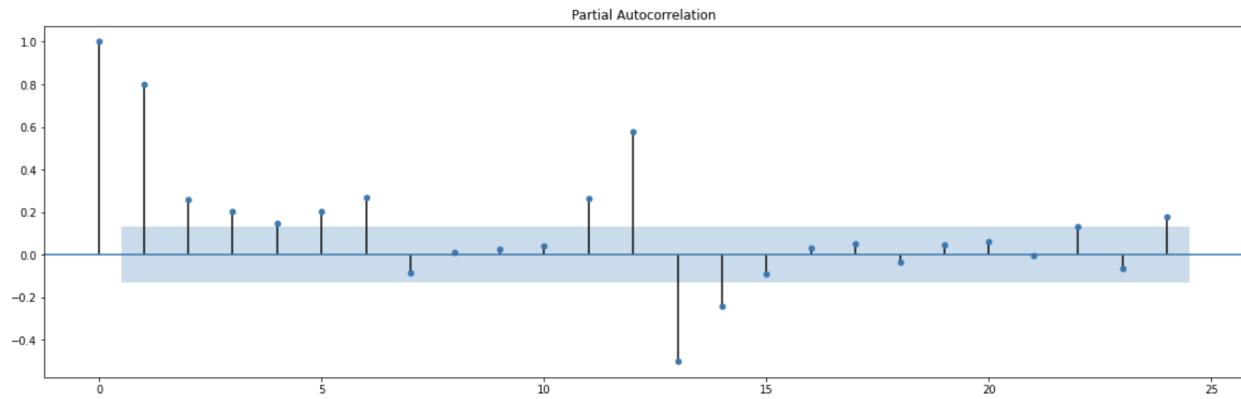
Partial Autocorrelation

- This is similar to AutoCorrelation with only a small difference.
- We try to find a relationship between the original time series $y(t)$ and time series lagged with i steps $y^i(t)$, where $i=1,2,3,\dots$, to find the optimal value of m .
- The difference is that: **All intermediate/indirect correlations are removed.**
 - The correlation between observations at successive time steps is a linear function of the indirect correlations.
 - These indirect connections are eliminated using the **partial autocorrelation function (PACF)**.

For example

- When we're considering the correlation between $y(t)$ and say, $y^{12}(t)$,
 - Then, we do not want this correlation value to get corrupted by the correlations when $i=1,2,3,\dots,11$ (i.e. the intermediate correlations)
- The partial correlation for each lag is the **unique correlation** between the two observations after the intermediate correlations have been removed.
- This is also known as **Conditional AutoCorrelation**.

PACF Plot:



- Here, one can see:
 - Value at $i=1$ is high: This means that given the values of time series $y(t)$, we can compute the values of a time series with lag 1, $y^1(t)$ with ease.
 - There is a high value at $i=12$

- This means that even if we ignore the information given by all the time series with $lags=1,2,\dots,11$, the information carried in time series $y^{12}(t)$ alone is very high.
- This shows strong seasonality

ARIMA Family of Forecasting Techniques

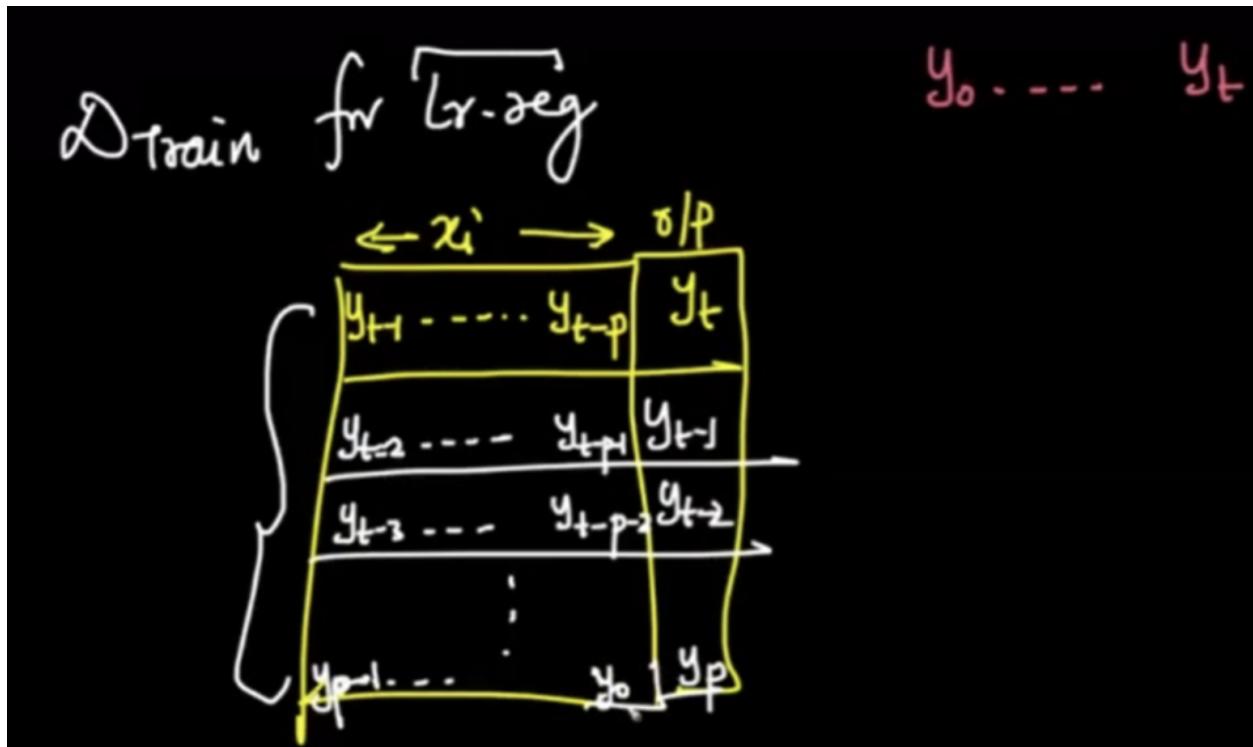
Auto Regression (AR)

- For stationary time series, one is helpless because of the absence of pattern in the series.
- Any trend or seasonality that was present has been removed from the series, and it will be added back later, in the final prediction.
- For such cases we study a new family of models called **ARIMA**.
- Though such series look like they are completely random, there is still some extent of forecastability here, there is still information left to be extracted from stationary series.

What if we had a feature in our time series, besides the value to be predicted?

- In that case, one could just utilize Linear Regression, by mapping this feature's values with the value to be forecasted.
- For creating a new feature, we can map the value of the stationary time series at time t with the value of series at time $t-1, t-2, t-3, \dots, t-p$, where p could be a **hyperparameter** we set.
- This way, now our data becomes as shown.
 - It contains date as an index,
 - Past values $y_{t-1}, y_{t-2}, \dots, y_{t-p}$ as features and
 - value at time t as the value to be predicted (\hat{y}_t)

Now we can successfully implement Linear Regression using these features.



- Since the aim is to convert the forecasting problem to Linear Regression, what we're doing is;
 - Future value $\hat{y}_t = \text{LinearRegression}(\text{Past } p \text{ values})$

- Thus, the forecasting problem is converted into the following form:

$$\hat{y}_t = \alpha_0 + \alpha_1 y_{t-1} + \alpha_2 y_{t-2} \dots + \alpha_p y_{t-p} + \epsilon_t$$

where;

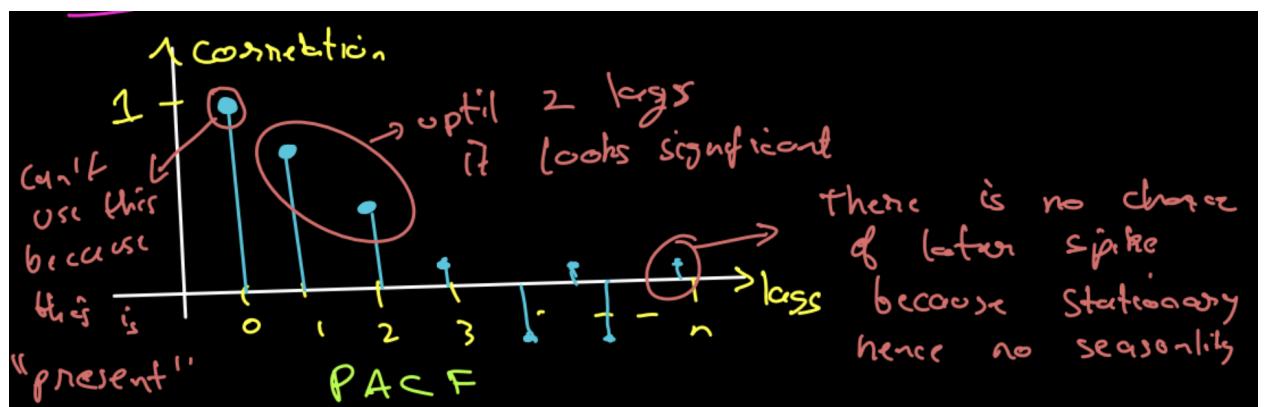
- α_0 : intercept (parameter)
- ϵ_t : error term
- $\alpha_1, \alpha_2, \dots, \alpha_p$: weights (parameters)
- p : No of past values to be considered (hyperparameter)

- This seems similar to Simple Exponential Smoothing (SES).
- Though we are essentially doing a **weighted average** of the past time series values in both SES and AR models, there is a fundamental difference.
 - In the case of SES,

- The weights are **exponentially decaying**
- The hyperparameter is **α**
- In the case of AR,
 - The weights are **learned** by multiple iterations.
 - The hyperparameter is **p**
- **Pre-requisites of the AR model:**
 - The idea is that this assumption will be true if the Partial Auto Correlation Plot has a high value at lag = k.
 - This way, from the plot we know that the future value is highly correlated with one value in the past, which means that it makes sense for us to compute linear regression on past 1 value.
 - We are using PACF because we don't want features to be correlated with each other in LR. PACF can help to identify that

Deciding the value of p

- We look at the PACF plot, and based on the values there, we decide how many lag values we can consider.
- For example, in the given plot,
 - we consider 2 lags only,
 - as for the third lag, the PACF value is 0.1, so considering it will not give as promising results.



Moving Averages (MA)

- The idea here is to use the value of the series at time t-1, we use the error of value at t-1 from the mean of data in the regression setup
- The error of each data point in the series from the mean/average, would be different.
- This should also work, as we are able to successfully create a new feature that is unique for each point.
- In fact, this idea is called the **Moving Averages (MA)** technique.
- Though the name is the same as the smoothing technique, this has nothing to do with that and is a completely different concept.
- The formulation of MA is as follows:

$$\hat{y}_t = \mu + m e_{t-1}$$

where,

- μ -> mean of entire data
- e_{t-1} -> error of the data point at time t-1, with respect to the mean
- m -> constant

- This idea can be extended to the order of q. In that case, the formulation becomes:

$$\hat{y}_t = \mu + m_1 e_{t-1} + m_2 e_{t-2} \dots + m_q e_{t-q} + \epsilon_t$$

Here, ϵ represents the final error remaining that is actually truly random, which we cannot help. This is also added for representation.

NOTE:

- q becomes the **hyperparameter** for the MA(q) model
- In the case of MA, there is a fixed way to determine the value of q, we need to try a bunch of different values to find the best fit.

Auto Regression - Moving Averages (ARMA)

- The combined technique of Auto Regression (AR) and Moving Averages (MA) is called the ARMA model.
- While combining the two ideas, p : order of AR and q : order of MA, p may or may not be equal to q
- $\alpha_1, \alpha_2, \dots, \alpha_p$: coefficients of AR
- m_1, m_2, \dots, m_q : coefficients of MA
- Hence the formulation becomes:

$$\hat{y}_t = c + \alpha_1 y_{t-1} + \dots + \alpha_p y_{t-p} + m_1 e_{t-1} + \dots + m_q e_{t-q} + \epsilon_t$$

i.e. $\hat{y}_t = c + \epsilon_t + \sum_{i=1}^p \alpha_i y_{t-i} + \sum_{j=1}^q m_j e_{t-j}$

- Here, p and q are hyperparameters. Thus it is also called **ARMA(p,q)** model.
- The major limitation of this technique is that it cannot handle non-stationary time series because, if we're training a Linear Regression, the variables can not be dependent on each other.

ARIMA

- If a time series is not stationary,
 - We perform **differencing** to de-trend
 - Then we apply the ARMA technique, to get an approximation.
 - Now to get a good forecast, we need to **integrate** the trend **back** to get the final result
- This technique is called **ARIMA**
- Instead of us manually doing integrations (and note that it can get very hard when you need to double / triple differentiate), we can simply use the ARIMA model, which does this job for us.

Formulation of ARIMA

- ARIMA can be formulated as a summation of
 - Differencing of order d
 - $d=1$ for linear trends, $d=2$ for quadratic trends, ...
 - Auto Regression of order p
 - Moving Averages of order q
 - Getting the resulting forecasts

- Integrating d times to restore the trend back

$$\hat{y}'_t = C + a_1 y_{t-1} + a_2 y_{t-2} + \dots + a_n y_{t-p} + m_1 e_{t-1} + m_2 e_{t-2} + \dots + m_d e_{t-d}$$

$$\hat{y}_t = \underbrace{y_{t-1} + \hat{y}'_t}_{\text{Many additions over time}} : \text{if } d=1$$

will result in integration

$$\hat{y}_t = y_{t-1} + \hat{y}'_{t-1} + \hat{y}''_t : \text{if } d=2$$

analogous to double integration

SARIMA

- If we wish to account for seasonality, the process becomes:-
 - differentiating ($x[i] - x[i-T]$), to remove seasonality
 - perform AR and MA
 - integrating the seasonality back
- This is very tiring.
- Instead of doing so much work to utilize the ARIMA model, we can just apply another model called the **SARIMA model** directly.
- There are 7 parameters for SARIMA: P, Q, D, p, q, d, s

Hyperparameter s

- The parameter s represents **seasonality**
 - This value we can find using ACF and PACF plots.

- Alternately, we can treat s as a **hyperparameter** and tune it to get the best value.
- In case of lag of order 1, at time t, translates to y_{t-1}
- For example,
 - For a normal time series, a lag of order of 2 means y_{t-1} and y_{t-2}
 - And this makes sense as if we're in March 2022, then order of 2 means, lag of 1, i.e. February 2022 and lag of 2, i.e. January 2022
 - In case of yearly seasonality, if we're in March 2022, a lag of order 2, means March 2021, and March 2020.
 - Therefore, here, an order of m gets translated as $y_{t-s}, y_{t-2s}, \dots, y_{t-ms}$.
- There is one **problem** with the SARIMA Model in terms of capturing seasonality.
- **We can only use one value of seasonality.**

Hyperparameter P

- When we set the order of AR as p, the formulation becomes:
$$\hat{y}_t = \alpha_0 + \sum_{i=1}^p \alpha_i y_{t-i} + \sum_{i=1}^Q c_i \epsilon_{t-is}$$
- **The effect of setting the 'P' hyperparameter for SARIMA is:**
 - Since our seasonality is 12
 - Essentially, P enables in creating of an AR Model on data that is 12 months old, 24 months old, ..., 12P months old: $y_{t-12}, y_{t-24}, \dots, y_{t-12P}$.
 - So this is exactly like an AutoRegression, but with seasonality.
- Suppose we wish to forecast \hat{y}_{100} (i.e. t=100) with the following hyperparameter values:-
- $p=4$
 - Contribution to final forecast:

$$\alpha_1 y_{99} + \alpha_2 y_{98} + \alpha_3 y_{97} + \alpha_4 y_{96}$$

- s=12
- P=3
 - Contribution to final forecast:

$$\beta_1 y_{100-12=88} + \beta_2 y_{88-12=76} + \beta_3 y_{76-12=64}$$

Hyperparameter Q

- Similarly, the effect of setting the Q hyperparameter for SARIMA is:

$$\sum_{i=1}^Q c_i \epsilon_{t-is}$$

- Suppose if seasonality in a certain time series data is 12.
- Essentially, Q enables the creation of an MA Model on data that is 12 mon old, 24 months old, ..., 12Q months old: $y_{t-12}, y_{t-24}, \dots, y_{t-12Q}$.
- So this is exactly like a Moving Average, but with seasonality.
- Suppose we wish to forecast \hat{y}_{100} with the following hyperparameter values:-
 - q = 4
 - Contribution to final forecast:

$$m_1 \epsilon_{99} + m_2 \epsilon_{98} + m_3 \epsilon_{97} + m_4 \epsilon_{96}$$

- s = 12
- Q = 3
 - Contribution to final forecast:

$$c_1 \epsilon_{100-12=88} + c_2 \epsilon_{88-12=76} + c_3 \epsilon_{76-12=64}$$

Hyperparameter D

- Recall that hyperparameter d performs differencing on the time series d times, before applying the model.
 - For example if d=1: $y'_t = y_t - y_{t-1}$

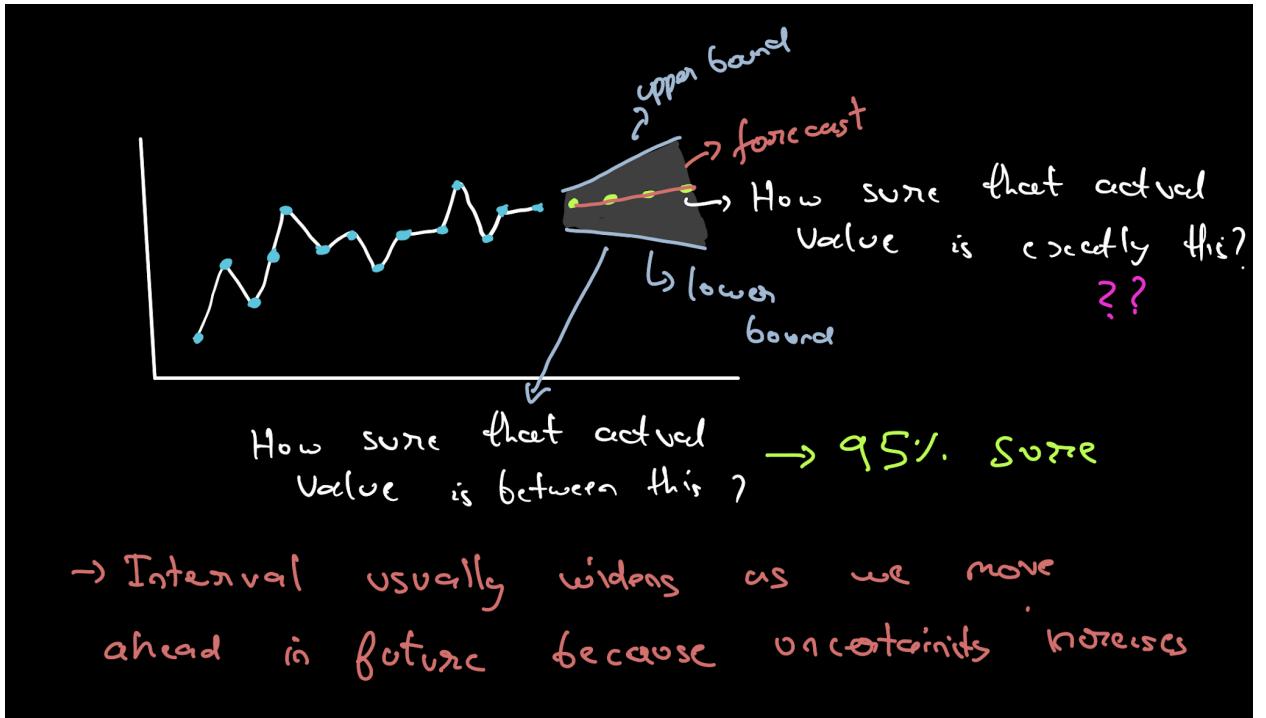
- Similarly, D helps in doing **seasonal differencing** on the time series.
 - For example if $D=1$: $y'_t = y_t - y_{t-1}$
 - Since $s = 12$: $y'_t = y_t - y_{t-12}$

So, to sum it up SARIMA can be formulated as

$$\hat{y}_{t+1} = p(\text{AR term}) + q(\text{MA term}) + d(\text{differencing}) + D(\text{differencing}) + P(\text{AR-Seasonality}) + Q(\text{MA - Seasonality})$$

Confidence Intervals

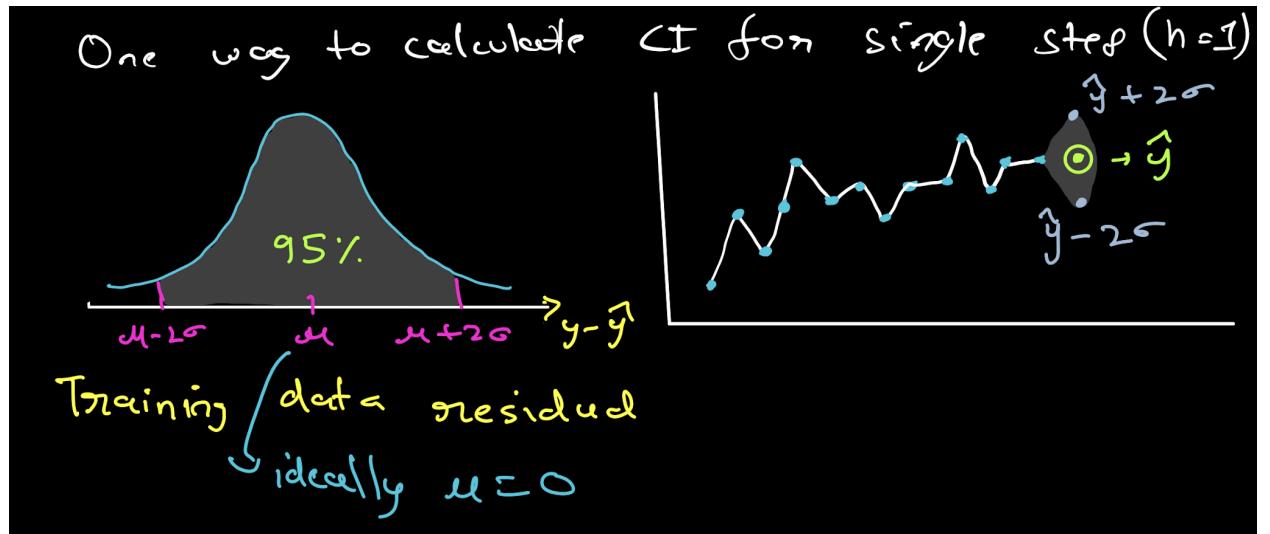
- In addition to making predictions, the models can also provide Confidence Intervals.
- In addition to making predictions, time series models can also provide Confidence Intervals.
- Owing to the **potential error** of our forecast, and the **unpredictability of the future**, we cannot claim to be a hundred percent sure of our forecasts.
- But we can still, based on our study of time series, say that the future value would lie in the **range** of $[A, B]$.
- Naturally A becomes the minimum possible value of our target variable at time t, and B becomes the maximum possible value at time t.
- This $[A, B]$ becomes a range for determining the **real possible value** for a certain prediction.
- It is called the **confidence interval**.
- It can also be used for performance evaluation.



- Confidence Intervals are required for determining the uncertainty of your predictions, after making forecasts.
- Confidence Intervals provide an expected range for the real observation. When making decisions, one can utilize the range of information better, than just a single point.
- Different algorithms have different formulations for CI.
- These can be complicated to derive and are out of scope for this lecture.
- We can get the upper and lower values according to the models available in the statsmodel library.
- It takes a value of a parameter: **alpha**, which is a measure of how much confidence we want in our forecasts.
- One example of a 1-step ahead forecast CI is to just take the residual distribution.
- For example, the 95% confidence interval will be

$$\hat{y}_{t+1} = \hat{y}_t + u_{\text{resid}} \pm 2\sigma_{\text{resid}}$$

- Note that for a good forecast U_{resid} should be close to 0



- To interpret confidence intervals, let our forecasted value is \hat{y} . And you pass a value of **alpha = 0.05** for which you get the confidence interval as (m,n) .
- It means that the time series model will estimate the upper(n) and lower(m) bound of values around the forecast, where there is only a 5% chance that the real value will **not be** in that range.
- That is, 95% that our forecast will fall within the range (m,n) .

SARIMAX Model

- We can utilize the exogenous variable and incorporate it in our SARIMA Model.
- This idea of incorporating exogenous variables into SARIMA gave rise to a new model: **SARIMAX Model**
- Here, the X represents an exogenous variable
- The only difference between SARIMA and SARIMAX is that here, we can incorporate exogenous variables into the calculations of our forecasts.
- Exogenous variables are assigned a weight, say w_i .
- We don't need to initialize this, it is learned and trained by SARIMAX, and taken care of under the hood.
- This is done in addition to the SARIMAX operations.

- SARIMAX model takes exogenous variables into account
 - i.e. variables measured at time t that influence the value of our time series at time t , but that is not auto regressed on.

- To do this, we simply add the terms on the right-hand side of our ARIMA and SARIMA equations.

Since we have not seen eqn for SARIMA,
lets build intuition using ARMA

ARMA

$$\hat{y}_{t+1} = \alpha_1 y_t + \alpha_2 y_{t-1} + \dots + \alpha_p y_{t-p} + b_1 e_t + b_2 e_{t-1} + \dots + b_q e_{t-q} + a_0 + e$$

ARMA (x)

$$\hat{y}_{t+1} = \alpha_1 y_t + \alpha_2 y_{t-1} + \dots + \alpha_p y_{t-p} + b_1 e_t + b_2 e_{t-1} + \dots + b_q e_{t-q} + \underbrace{c_1 x_1 + c_2 x_2 + \dots + c_n}_\text{exogenous var} + a_0 + e$$

Similar change to be made for SARIMA

Facebook's Prophet

- One of the drawbacks of the SARIMAX Model is that one cannot have multiple seasonality and can only select one value.
- To overcome this disability, we're studying an open-source tool/library called prophet which is developed by Facebook

Following are the features of Facebook's Prophet:

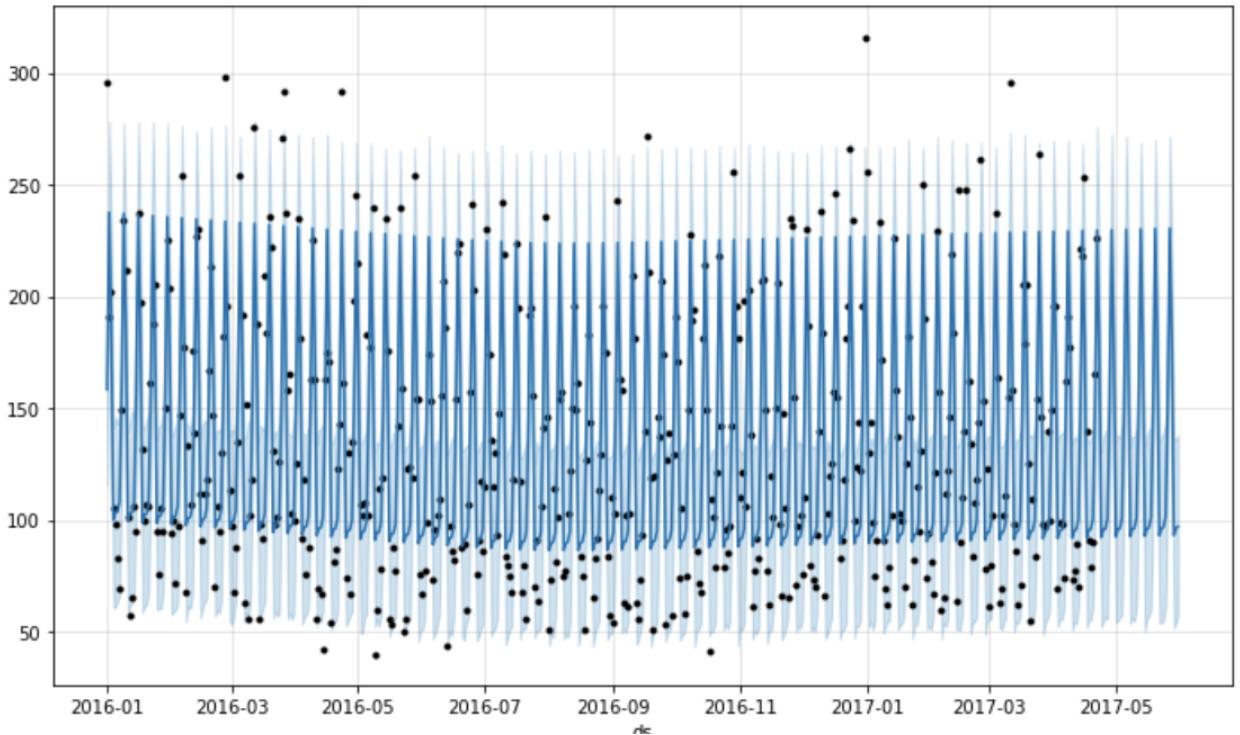
- Provides intuitive parameters which can be **easily tuned**
- It is **robust to missing data and shifts in the trend** and typically **handles outliers** well.
- It can account for **multiple seasonalities**. This is possible because, under the hood, the math of seasonalities is based on **Fourier transforms**, which help incorporate this.
- The Prophet uses a decomposable time series model with three main model components
- They are combined in the following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon t$$

- **$g(t)$** : piecewise linear or logistic growth curve for modeling non-periodic changes in time series (**trend**)
 - **$s(t)$** : periodic changes (e.g. weekly/yearly **seasonality**)
 - **$h(t)$** : effects of **holidays** (user provided) with irregular schedules
 - **ϵ_t** : **error term** accounts for any unusual changes not accommodated by the model.
- For using Prophet, the data set should contain only two columns with column names as ‘ds’ and ‘y’
 - ‘ds’ should always be in ‘date-time’ format. And ‘y’ represents a feature that you want to forecast.
 - To install prophet to your environment, you can use the following environments:

```
!pip install pystan~=2.14
!pip install fbprophet
```

```
→ from fbprophet import Prophet
m = Prophet()
m.fit(df[['ds', 'y']][:-39]) #here we are leaving
last 39 observations because we will predict it in
'future'
future = m.make_future_dataframe(periods=39, freq="D")
forecast = m.predict(future)
fig = m.plot(forecast)
```



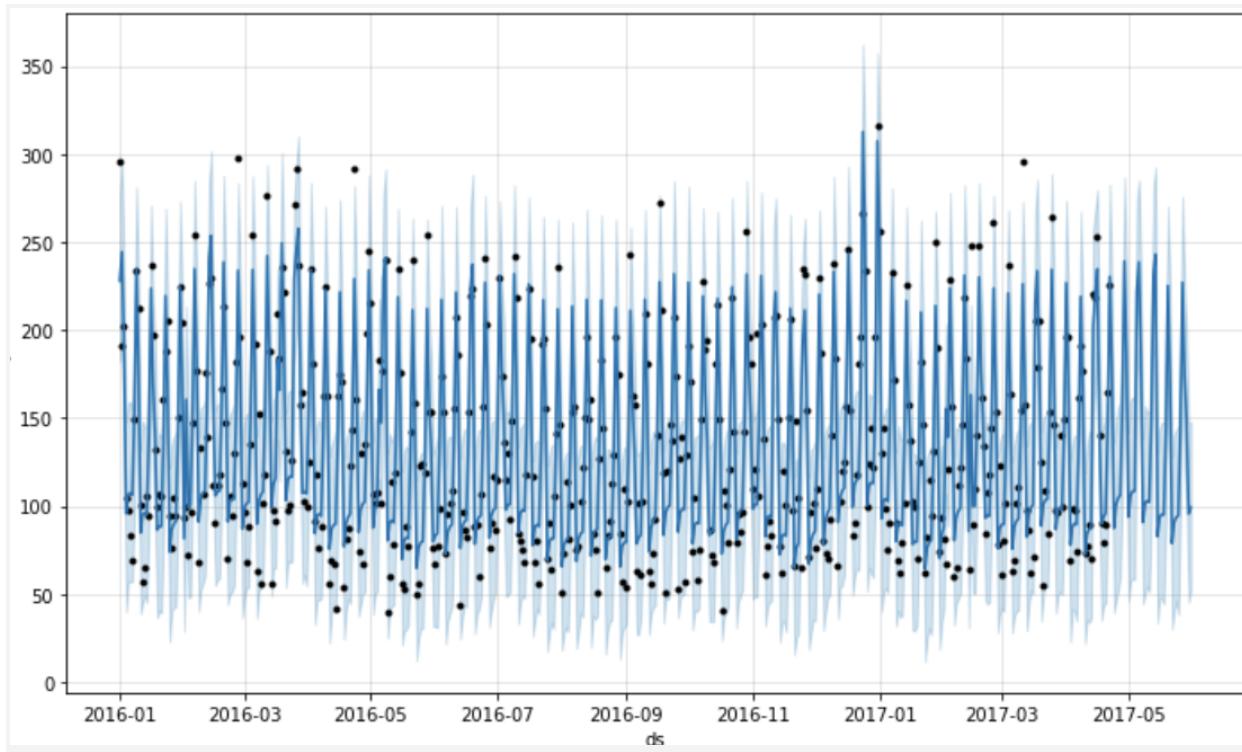
- In the plot, black dots are actual visits, deep blue lines are the predicted visits and light blue lines are the 95% confidence interval around the prediction.
 - You can see that the lines are flat and the model is not able to capture the seasonality properly so it is not a good fit.
 - Here light blue lines are 95% confidence intervals around the predictions.
 - Here we also didn't do anything explicitly for Nan values it was handled by the prophet.
 - We can use '`add_regressor`' for adding features to the prophet model.
 - There is another interesting parameter here: '`changepoint_prior_scale`'
 - If the trend changes are overfitting (too much flexibility) or underfitting (very less flexibility)
 - We can adjust the strength of sparse prior using this parameter.
 - Default value: 0.05
 - Increasing this will make the trend more flexible.

```

→      model2=Prophet(interval_width=0.95,
yearly_seasonality=True,
weekly_seasonality=True, changepoint_prior_scale=4)

model2.add_regressor('holiday') #adding holidays data in
the model3
model2.fit(df[:-39])
forecast2 = model2.predict(df)
fig = model2.plot(forecast2)

```



Changing Trends

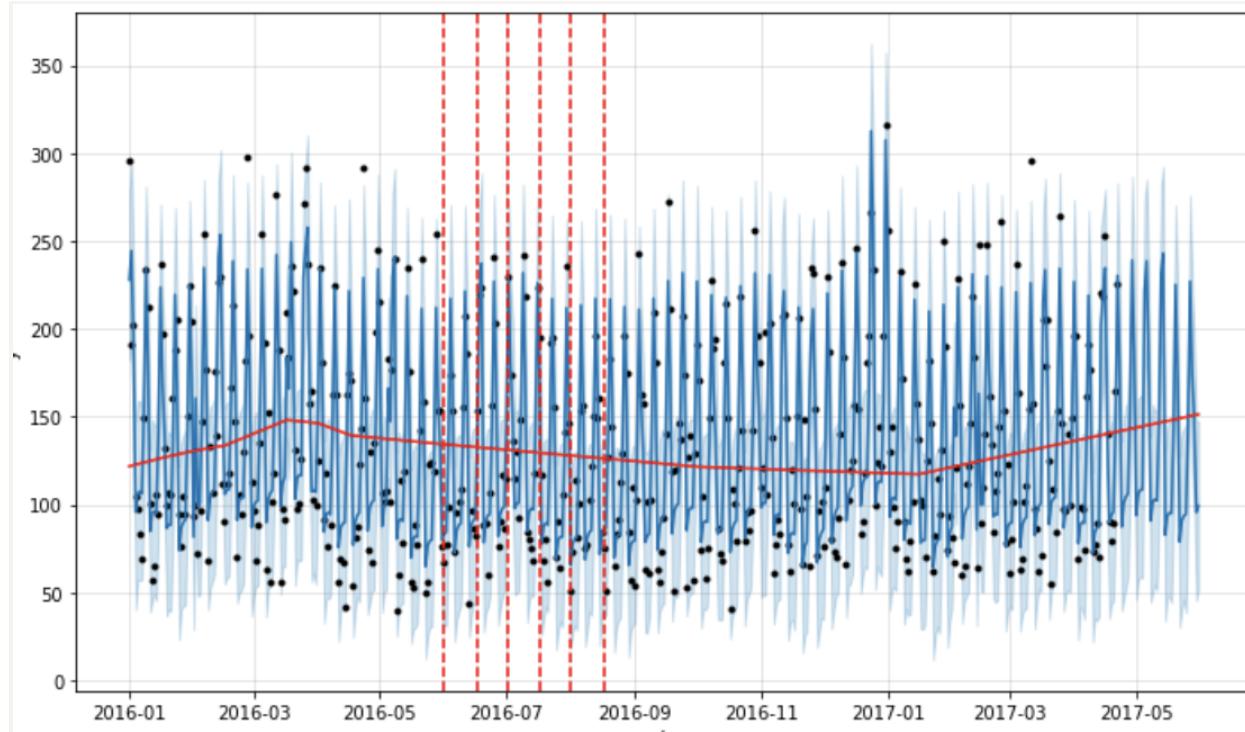
- Prophet automatically detects the changepoints and will allow the trend to adapt appropriately.
- Prophet detects changepoints by first specifying a large number of potential changepoints at which the rate is allowed to change.

```
→      from fbprophet.plot import add_changepoints_to_plot
```

```

fig = m.plot(forecast2)
a = add_changepoints_to_plot(fig.gca(), m, forecast2)

```



- The vertical lines in this figure indicate where the potential change-points were placed.
- Facebook prophet provides automated methods to forecast.
- The model has easily interpretable parameters that can be changed by the analyst to impose assumptions on the forecast.

Benefits of Prophet

- Prophet is a simple library and is great for beginners.
- It works best with time series that have strong seasonal effects.
- Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.
- We can add multiple regressors or exogenous variables.
- It forms a very good baseline model because almost no feature engineering is required.
- Interpretability is one of the key advantages of the Prophet.

- If your time series follows some business cycles, you can obtain very decent performance quickly.
- It can also be helpful while detecting change points.