K-Means

1) Data Preprocessing :

The K-Means clustering algorithm has fundamental mathematical assumptions that make preprocessing not just beneficial, but essential for obtaining accurate and meaningful results. The core of the issue lies in the algorithm's reliance on the Euclidean Distance metric.

Why does KMeans need normal data ?

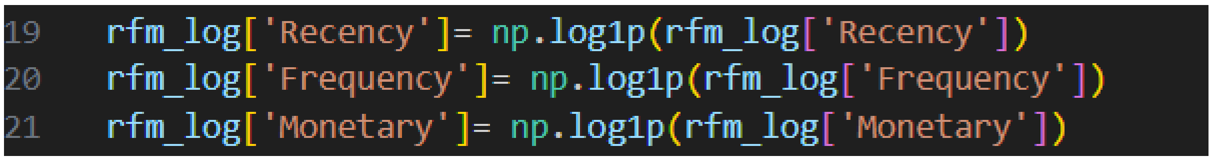
General principle : K-Means assigns data points to clusters based on their proximity to centroids, calculated using Euclidean distance.
⇒ In a typical RFM dataset, the distributions of **Frequency** and **Monetary** values are highly right-skewed ⇒ This skewness creates a major problem for the Euclidean distance calculation.

Data Transformation : Logarithmic method !

To mitigate the skewness and make the data more suitable for K-Means, we applied a logarithmic transformation using np.log1p() ⇒ log(1 + x) instead of log(x) to handle 0 values, we cannot use log(0) so log(1+0)=0.

The log function compresses the long tail of the distribution and expands the scale for the majority of customers with lower values ⇒ this build a distribution more symmetric and closer to Gaussian distribution

⇒ After the log transformation, the Frequency and Monetary features no longer dominate the distance metric. All three features (Recency, Frequency, Monetary) now contribute more equally to determining which customers are similar.



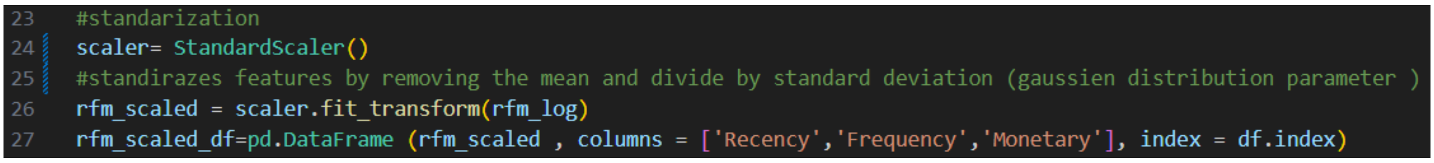
```
rfm_log['Recency']= np.log1p(rfm_log['Recency'])
rfm_log['Frequency']= np.log1p(rfm_log['Frequency'])
rfm_log['Monetary']= np.log1p(rfm_log['Monetary'])
```

Standardization : Z-Score Normalization

Even after the log transformation, the features might still be on different scales
⇒ So we need to normalize values to put them in the same range

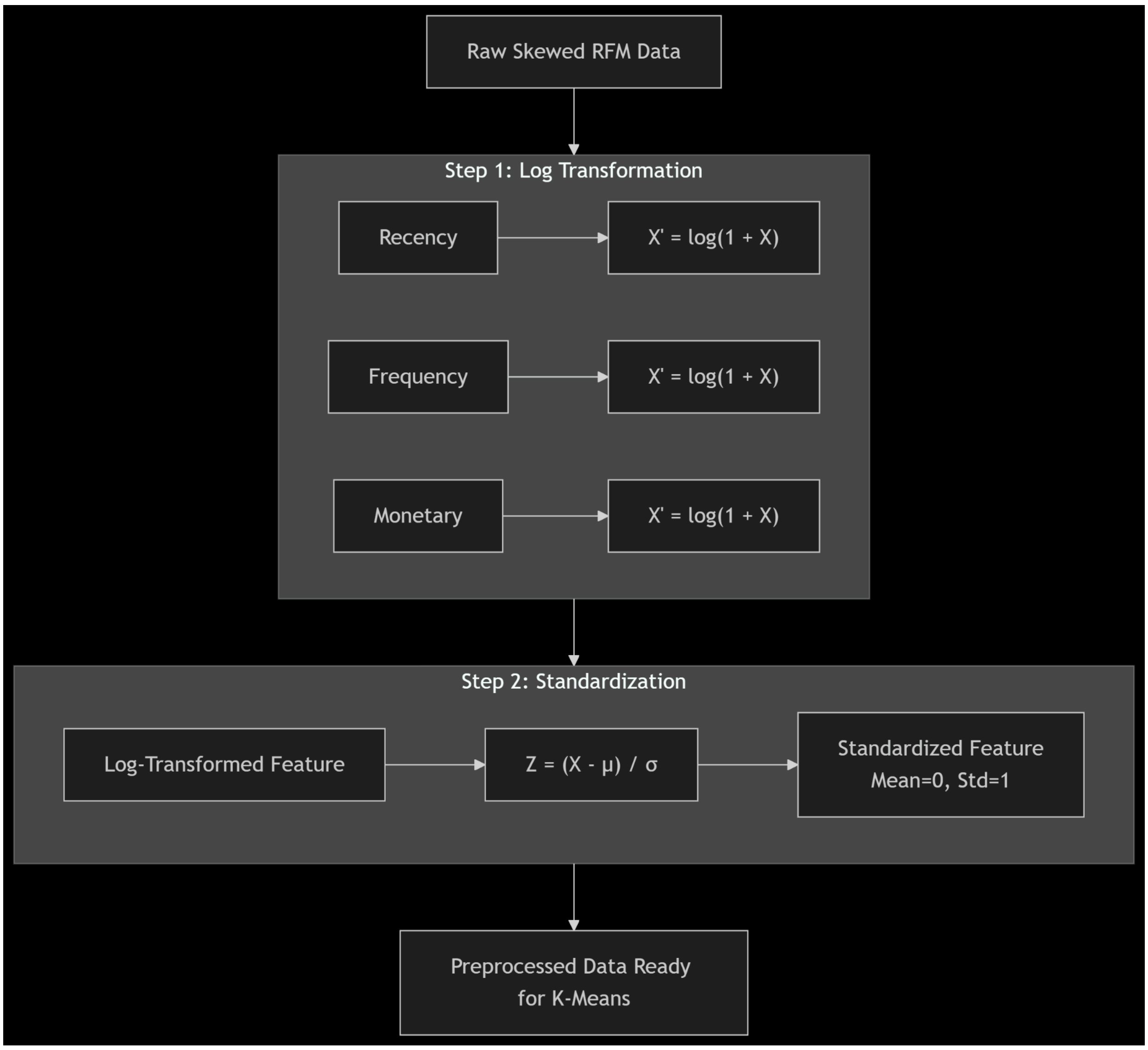
$$z = (x - \mu)/\sigma$$

x: original value
μ: mean of the feature
σ: standard deviation of the feature



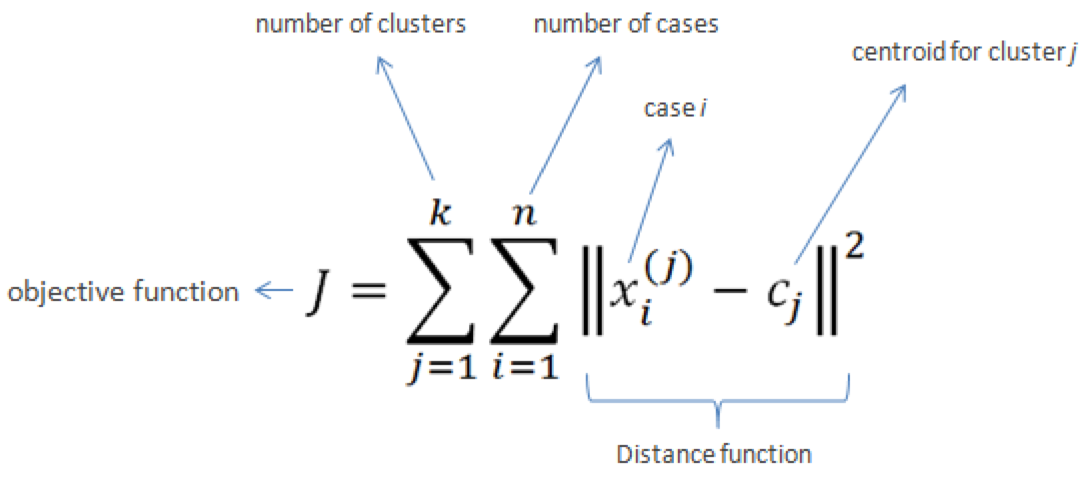
```
#standarization
scaler= StandardScaler()
#standirazes features by removing the mean and divide by standard deviation (gaussien distribution parameter )
rfm_scaled = scaler.fit_transform(rfm_log)
rfm_scaled_df=pd.DataFrame (rfm_scaled , columns = ['Recency','Frequency','Monetary'], index = df.index)
```

Data Preprocessing Map with necessary steps :



KMeans Algorithm :

The entire algorithm is designed to minimize the following objective function:(Inertia)



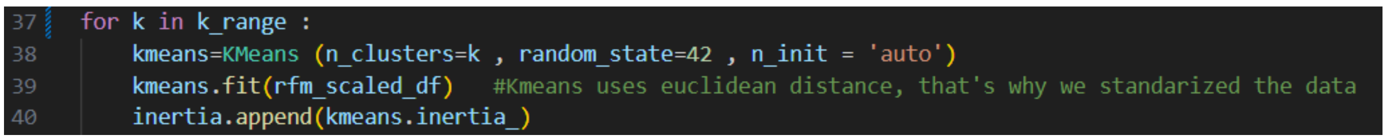
$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \underbrace{\left\| x_i^{(j)} - c_j \right\|^2}_{\text{Distance function}}$$

The algorithm's steps are all in service of reducing the value of J, which represents the total squared error of all points from their respective cluster centers (centroides)

NB: Euclidean distance Formula:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

⇒ Then, we hold the centroids μ fixed and assign each data point x to the cluster whose centroid is closest ⇒ This is done by calculating the Euclidean distance from the point to every centroid and finding the minimum.



```
for k in k_range :
    kmeans=KMeans (n_clusters=k , random_state=42 , n_init = 'auto')
    kmeans.fit(rfm_scaled_df)    #Kmeans uses euclidean distance, that's why we standarized the data
    inertia.append(kmeans.inertia_)
```

Next Step:

⇒ Update: Recalculate centroids as the mean of points in each cluster

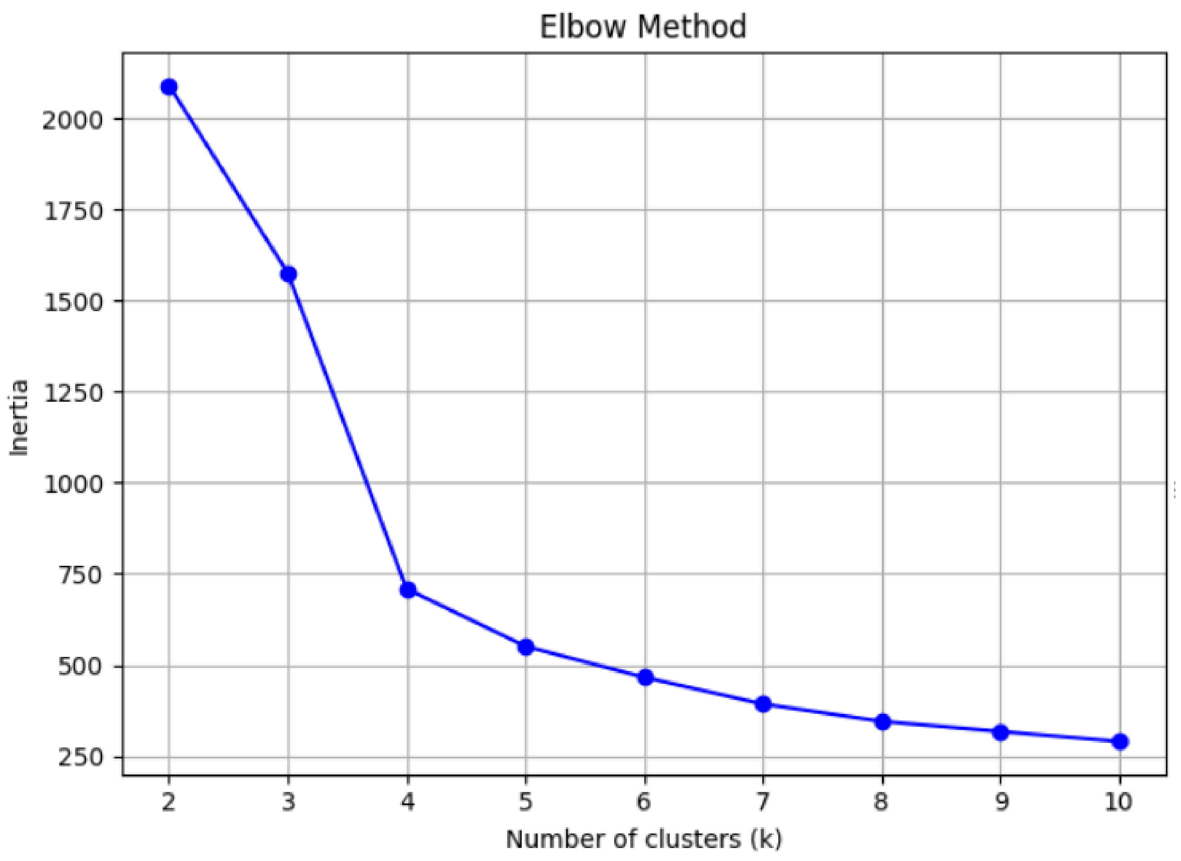
NB: Mean Formula:

$$\mu = \frac{\sum x}{n}$$

How to Conclude the Optimal Number of Clusters ?

Elbow Method (Inertia)

Inertia is the K-Means objective function itself
⇒ Lower inertia means tighter clusters
⇒ As k increases, inertia will always decrease
⇒ The goal is to find the point where the rate of decrease sharply changes, forming an "elbow".

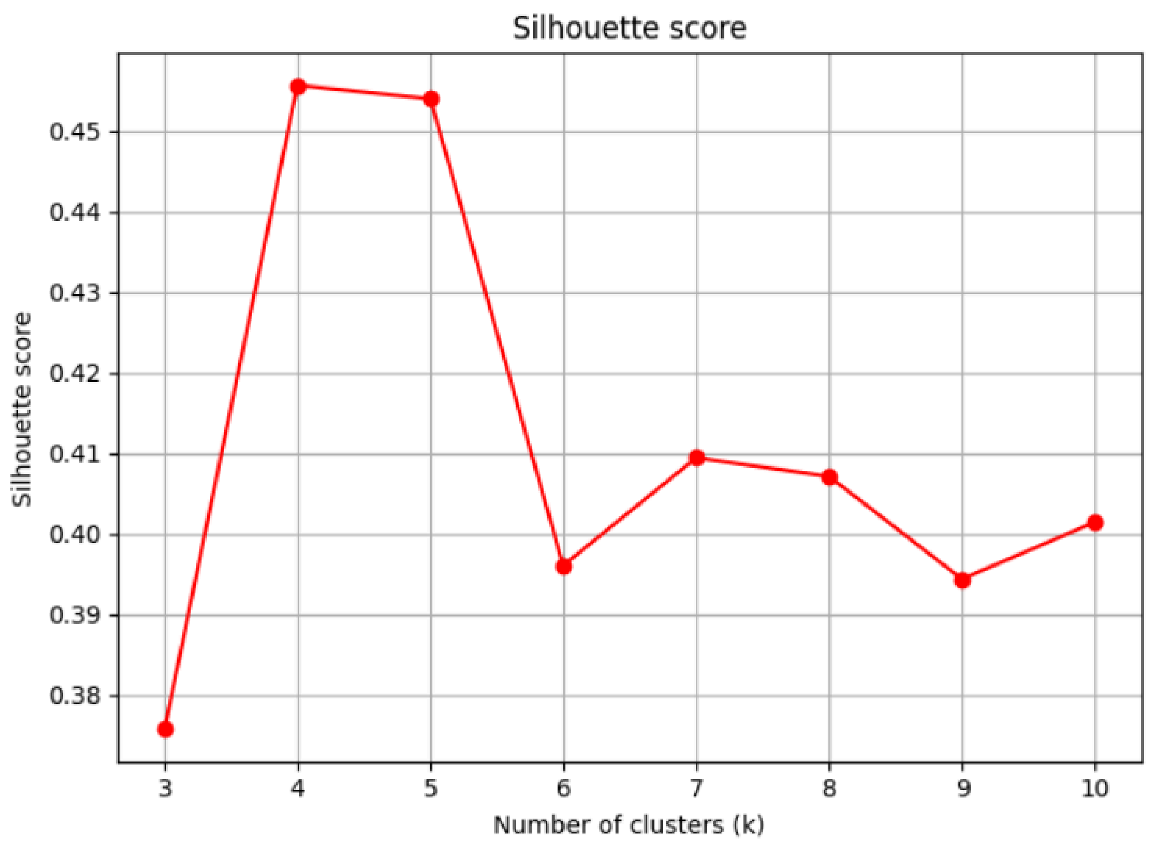


Silhouette Score :

measures the quality of the clustering, based on how well-separated the clusters are. It ranges from <u>-1 to 1</u>.

- A score close to 1 means clusters are dense and well-separated.
- A score around 0 means clusters are overlapping.
- A negative score indicates that samples might have been assigned to the wrong cluster.

⇒ The optimal k is the one that maximizes the Silhouette Score.



How to choose K ?

-The ideal scenario is where the "elbow" in the inertia plot aligns with a peak in the silhouette score plot
-The silhouette score is often a more reliable metric because it directly measures clustering quality and separation

```
#the elbow (the point where the curve bends ) indicates the optimal K
#the K with the highest silhouette score is usually the best choice
optimal_k = 4
```