K-Means Clustering Algorithm

K-Means is a popular unsupervised machine learning algorithm used for **clustering** data into **K** distinct groups (or clusters) based on feature similarity. The goal of K-Means is to partition the data into K clusters where each data point belongs to the cluster with the nearest mean (centroid).

How K-Means Clustering Works

The algorithm operates iteratively and involves the following steps:

1. **Initialization:**
   - Select the number of clusters, **K**.
   - Randomly initialize **K centroids** (points representing the center of each cluster) within the data space.
2. **Assignment Step (E-Step):**
   - Each data point is assigned to the nearest centroid based on a distance metric (typically Euclidean distance).
   - This creates **K clusters**, where each cluster contains points that are closest to its centroid.
3. **Update Step (M-Step):**
   - After assigning the data points to the clusters, the centroids are recalculated by finding the **mean** of all the data points in each cluster.
   - The new centroids are the updated "center" of the clusters.
4. **Iteration:**
   - Steps 2 and 3 are repeated iteratively until convergence. Convergence occurs when the centroids stop changing significantly or when a maximum number of iterations is reached.
5. **Final Clusters:**
   - After convergence, the algorithm outputs **K clusters**, with each point belonging to one of the clusters based on proximity to the centroid.

Distance Calculation in K-Means

The most commonly used distance metric in K-Means clustering is the **Euclidean distance**. The Euclidean distance between two points (x) and (y) in an n-dimensional space is calculated using the formula:

$$d(x, y) = \sqrt{((x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2)}$$

Where:

- (x) and (y) are two data points with coordinates ($(x_1, x_2, ..., x_n)$) and ($(y_1, y_2, ..., y_n)$) in n-dimensional space.
- The distance is the square root of the sum of squared differences between corresponding dimensions of the points.

This distance metric is used during the **Assignment Step** to assign each point to the nearest centroid.

Example of K-Means Clustering

Let's take a simple example with 2D data:

1. **Initialization:**
    - Suppose we have the following points: ((1, 2), (2, 3), (3, 4), (10, 11), (11, 12)).
    - Let's choose (K = 2) (i.e., we want 2 clusters).
    - Randomly select 2 initial centroids, say ((2, 3)) and ((11, 12)).
2. **Assignment Step:**
    - Calculate the distance from each point to both centroids.
    - Points closer to ((2, 3)) form one cluster, while points closer to ((11, 12)) form another cluster.
3. **Update Step:**
    - Compute the new centroid of each cluster by taking the mean of the points in each cluster.
    - If the new centroids are different from the previous centroids, update the centroids and repeat the assignment step.
4. **Iteration:**
    - The algorithm continues to iterate until the centroids no longer change, finalizing the clusters.

Advantages of K-Means

- **Simple and fast**: K-Means is computationally efficient and can scale to large datasets.
- **Easy to implement**: The algorithm is relatively simple to understand and implement.
- **Clusters with similar sizes**: K-Means tends to form clusters of similar size, making it suitable for problems where clusters have similar variance.

Disadvantages of K-Means

- **Sensitive to initialization**: Random initialization of centroids can lead to different clusters for different runs.
- **Fixed number of clusters**: The number of clusters, K, needs to be predefined, which is not always intuitive.

- **Sensitive to outliers**: Outliers can disproportionately affect the position of centroids.
- **Assumes spherical clusters**: The algorithm works well when clusters are roughly spherical and equally sized, but struggles with clusters of varying shapes and densities.

Distance Metrics in K-Means

Though Euclidean distance is the most common metric, other distance metrics can also be used depending on the nature of the data:

1. **Manhattan Distance**:
   - Formula:

$$d(x, y) = |x_1 - y_1| + |x_2 - y_2| + ... + |x_n - y_n|$$

   - Often used when the data is grid-like or for high-dimensional sparse data.
2. **Cosine Distance**:
   - Measures the cosine of the angle between two vectors, which can be useful when direction is more important than magnitude.
3. **Minkowski Distance**:
   - A generalization of both Euclidean and Manhattan distances.

In practice, **K-Means++** is a variant of K-Means that addresses the sensitivity to centroid initialization by choosing initial centroids in a smarter way, improving convergence.

Mathematics Behind K-Means Clustering

At the core of K-Means clustering, the goal is to minimize the **within-cluster sum of squares (WCSS)**, which measures the variability of the points within each cluster. The steps are as follows:

1. **Initial Centroid Selection**: Start by randomly selecting (K) centroids, where (K) is the number of clusters.
2. **Assignment Step**:
   - Each data point (x_i) is assigned to the closest centroid (\mu_j) using the Euclidean distance formula:

$$d(x_i, \mu_j) = \sqrt{\sum_{k=1}^{n} (x_{ik} - \mu_{jk})^2}$$

   Where:
   - (x_{ik}) is the (k)-th feature of the (i)-th data point.
   - (\mu_{jk}) is the (k)-th feature of the centroid of the (j)-th cluster.
3. **Update Step**:

- The centroid of each cluster is updated by calculating the **mean** of all data points assigned to that cluster:

$$\mu_j = \frac{1}{n_j} \sum_{i \in C_j} x_i$$

Where:

- (C_j) is the set of all points assigned to cluster (j).
- (n_j) is the number of points in cluster (j).

4. **Minimization of Objective Function**:

- K-Means minimizes the following **objective function**:

$$\text{WCSS} = \sum_{j=1}^{K} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

This represents the sum of squared distances between each point and its cluster centroid (within-cluster variance).

Stopping Criteria

K-Means algorithm iterates through the assignment and update steps until one of the following stopping criteria is met:

1. **Centroids do not change**: The centroids remain unchanged between two iterations.
2. **Convergence threshold**: The change in the positions of centroids between iterations is less than a predefined threshold.
3. **Maximum iterations**: A predefined maximum number of iterations is reached.

K-Means++

K-Means++ is an enhancement to the standard K-Means algorithm that improves the initial centroid selection. It avoids poor clustering due to random initialization by using the following steps:

1. **First Centroid**: Select the first centroid randomly from the data points.
2. **Subsequent Centroids**:
   - For each remaining centroid, compute the squared distance between each data point and its nearest existing centroid.
   - Select a new centroid with a probability proportional to the squared distance. This increases the likelihood of selecting centroids that are further apart from one another.

K-Means++ initialization results in faster convergence and often leads to better clustering solutions compared to random initialization.

Within-Cluster Sum of Squares (WCSS)

WCSS (Within-Cluster Sum of Squares) is a measure of the compactness of the clusters. It is computed as:

$$\text{WCSS} = \sum_{j=1}^{K} \sum_{i \in C_j} ||x_i - \mu_j||^2$$

Where $||x_i - \mu_j||^2$ is the squared distance between the data point x_i and its cluster centroid μ_j.

- **Lower WCSS** indicates better clustering since data points are closely grouped around their centroids.
- **Higher WCSS** implies greater spread within the clusters.

Elbow Method

The **Elbow Method** is a technique used to determine the optimal number of clusters (K). It involves:

1. **Run K-Means for various values of (K)**: Compute the WCSS for different numbers of clusters (typically (K = 1) to a higher number).
2. **Plot (K) vs. WCSS**: The x-axis represents the number of clusters (K), and the y-axis represents the WCSS.
3. **Identify the "Elbow"**: The "elbow" point on the plot is the value of (K) where the reduction in WCSS slows down significantly. This value of (K) is considered optimal.

Example of Elbow Method

Suppose we run K-Means with different values of (K) and obtain the following WCSS values:

- (K = 1): WCSS = 1000
- (K = 2): WCSS = 600
- (K = 3): WCSS = 400
- (K = 4): WCSS = 300
- (K = 5): WCSS = 250
- (K = 6): WCSS = 220

Plotting these values on a graph, we observe that the decrease in WCSS becomes smaller after (K = 3). Therefore, the "elbow" is at (K = 3), which might be the optimal number of clusters.

Key Takeaways

- **Mathematics**: K-Means clustering minimizes the WCSS by iteratively assigning points to clusters and updating centroids.
- **Stopping Criteria**: K-Means stops when centroids stop changing, a threshold is met, or a maximum number of iterations is reached.
- **K-Means++**: Improves the initialization of centroids, leading to faster convergence and better clusters.

- **WCSS**: Measures the compactness of the clusters; the lower the WCSS, the better.
- **Elbow Method**: Helps to determine the optimal number of clusters by identifying the point at which adding more clusters does not significantly improve the WCSS.

Import of required libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

Records of the given dataset

In [3]:
```
data = pd.read_csv("./data.csv", delimiter=',', encoding = "ISO-8859-1")
data.head(10)
```

Out[3]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerI
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	12/1/2010 8:26	7.65	17850.
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	12/1/2010 8:26	4.25	17850.
7	536366	22633	HAND WARMER UNION JACK	6	12/1/2010 8:28	1.85	17850.
8	536366	22632	HAND WARMER RED POLKA DOT	6	12/1/2010 8:28	1.85	17850.
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	12/1/2010 8:34	1.69	13047.

Information of the columns in a data

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

Description of numeric features

In [5]:
```
data.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%
Quantity	541909.0	9.552250	218.081158	-80995.00	1.00	3.00
UnitPrice	541909.0	4.611114	96.759853	-11062.06	1.25	2.08
CustomerID	406829.0	15287.690570	1713.600303	12346.00	13953.00	15152.00

Description of non numeric features

In [6]:
```
data.describe(exclude=['int64', 'float64']).T
```

Out[6]:

	count	unique	top	freq
InvoiceNo	541909	25900	573585	1114
StockCode	541909	4070	85123A	2313
Description	540455	4223	WHITE HANGING HEART T-LIGHT HOLDER	2369
InvoiceDate	541909	23260	10/31/2011 14:41	1114
Country	541909	38	United Kingdom	495478

Check for null values

In [7]:
```
data.isnull().sum()
```

Out[7]:
```
InvoiceNo          0
StockCode          0
Description     1454
Quantity           0
InvoiceDate        0
UnitPrice          0
CustomerID     135080
Country            0
dtype: int64
```

Shape of the available data

In [8]:
```
data.shape
```

Out[8]:
```
(541909, 8)
```

Delete all the missing records

In [9]:
```
data.dropna(inplace=True)
```

In [10]:
```
data.shape
```

Out[10]:
```
(406829, 8)
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

Checking of whether negative values present in Quantity and Unit Price

In [12]:
```
data[data['Quantity']<0]
```

Out[12]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Cus
141	C536379	D	Discount	-1	12/1/2010 9:41	27.50	
154	C536383	35004C	SET OF 3 COLOURED FLYING DUCKS	-1	12/1/2010 9:49	4.65	
235	C536391	22556	PLASTERS IN TIN CIRCUS PARADE	-12	12/1/2010 10:24	1.65	
236	C536391	21984	PACK OF 12 PINK PAISLEY TISSUES	-24	12/1/2010 10:24	0.29	
237	C536391	21983	PACK OF 12 BLUE PAISLEY TISSUES	-24	12/1/2010 10:24	0.29	
...	...	...	...	...	...	...	
540449	C581490	23144	ZINC T-LIGHT HOLDER STARS SMALL	-11	12/9/2011 9:57	0.83	
541541	C581499	M	Manual	-1	12/9/2011 10:28	224.69	
541715	C581568	21258	VICTORIAN SEWING BOX LARGE	-5	12/9/2011 11:57	10.95	
541716	C581569	84978	HANGING HEART JAR T-LIGHT HOLDER	-1	12/9/2011 11:58	1.25	
541717	C581569	20979	36 PENCILS TUBE RED RETROSPOT	-5	12/9/2011 11:58	1.25	

8905 rows × 8 columns

In [13]:
```
data[data['UnitPrice']<0].shape[0]
```

Out[13]:
```
0
```

Drop the negative values or quantity

In [14]:
```
data.drop(data[data['Quantity'] < 0].index, inplace=True)
```

In [15]:
```
data.shape
```

Out[15]: (397924, 8)

Calculation of total amount or monetary value

In [16]:
```
data['Sales'] = data['Quantity'] * data['UnitPrice']
new_data = data.groupby('CustomerID')['Sales'].sum().reset_index()
new_data
```

Out[16]:

	CustomerID	Sales
0	12346.0	77183.60
1	12347.0	4310.00
2	12348.0	1797.24
3	12349.0	1757.55
4	12350.0	334.40
...	...	...
4334	18280.0	180.60
4335	18281.0	80.82
4336	18282.0	178.05
4337	18283.0	2094.88
4338	18287.0	1837.28

4339 rows × 2 columns

In [17]:
```
type(new_data)
```

Out[17]: pandas.core.frame.DataFrame

Calculating the number of transactions of each customer

In [18]:
```
num_trans = data.groupby('CustomerID')['InvoiceNo'].count().reset_index()
num_trans
```

Out[18]:

	CustomerID	InvoiceNo
0	12346.0	1
1	12347.0	182
2	12348.0	31
3	12349.0	73
4	12350.0	17
...	...	...
4334	18280.0	10
4335	18281.0	7
4336	18282.0	12
4337	18283.0	756
4338	18287.0	70

4339 rows × 2 columns

In [19]:
```
type(num_trans)
```

Out[19]:
```
pandas.core.frame.DataFrame
```

Calculate Last Transaction

In [20]:
```
data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
```

In [21]:
```
data['InvoiceDate']
```

Out[21]:
```
0        2010-12-01 08:26:00
1        2010-12-01 08:26:00
2        2010-12-01 08:26:00
3        2010-12-01 08:26:00
4        2010-12-01 08:26:00
                 ...
541904   2011-12-09 12:50:00
541905   2011-12-09 12:50:00
541906   2011-12-09 12:50:00
541907   2011-12-09 12:50:00
541908   2011-12-09 12:50:00
Name: InvoiceDate, Length: 397924, dtype: datetime64[ns]
```

In [22]:
```
data['Last Transaction'] = (data['InvoiceDate'].max() - data['InvoiceDate
```

In [23]:
```
data['Last Transaction']
```

Out[23]:
```
0         373
1         373
2         373
3         373
4         373
          ...
541904      0
541905      0
541906      0
541907      0
541908      0
Name: Last Transaction, Length: 397924, dtype: int64
```

In [24]:
```
data.head(10)
```

Out[24]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerI
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.
5	536365	22752	SET 7 BABUSHKA NESTING BOXES	2	2010-12-01 08:26:00	7.65	17850.
6	536365	21730	GLASS STAR FROSTED T-LIGHT HOLDER	6	2010-12-01 08:26:00	4.25	17850.
7	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00	1.85	17850.
8	536366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:28:00	1.85	17850.
9	536367	84879	ASSORTED COLOUR BIRD ORNAMENT	32	2010-12-01 08:34:00	1.69	13047.

In [25]:
```
lt = data.groupby(['CustomerID','Country'])['Last Transaction'].max().res
```

In [26]:
```
lt
```

Out[26]:

	CustomerID	Country	Last Transaction
0	12346.0	United Kingdom	325
1	12347.0	Iceland	366
2	12348.0	Finland	357
3	12349.0	Italy	18
4	12350.0	Norway	309
...	...	...	...
4342	18280.0	United Kingdom	277
4343	18281.0	United Kingdom	180
4344	18282.0	United Kingdom	125
4345	18283.0	United Kingdom	336
4346	18287.0	United Kingdom	201

4347 rows × 3 columns

In [27]:
```
merge_table = pd.merge(lt, num_trans, how='inner', on='CustomerID')
new_df = pd.merge(merge_table, new_data, how='inner', on='CustomerID')
new_df
```

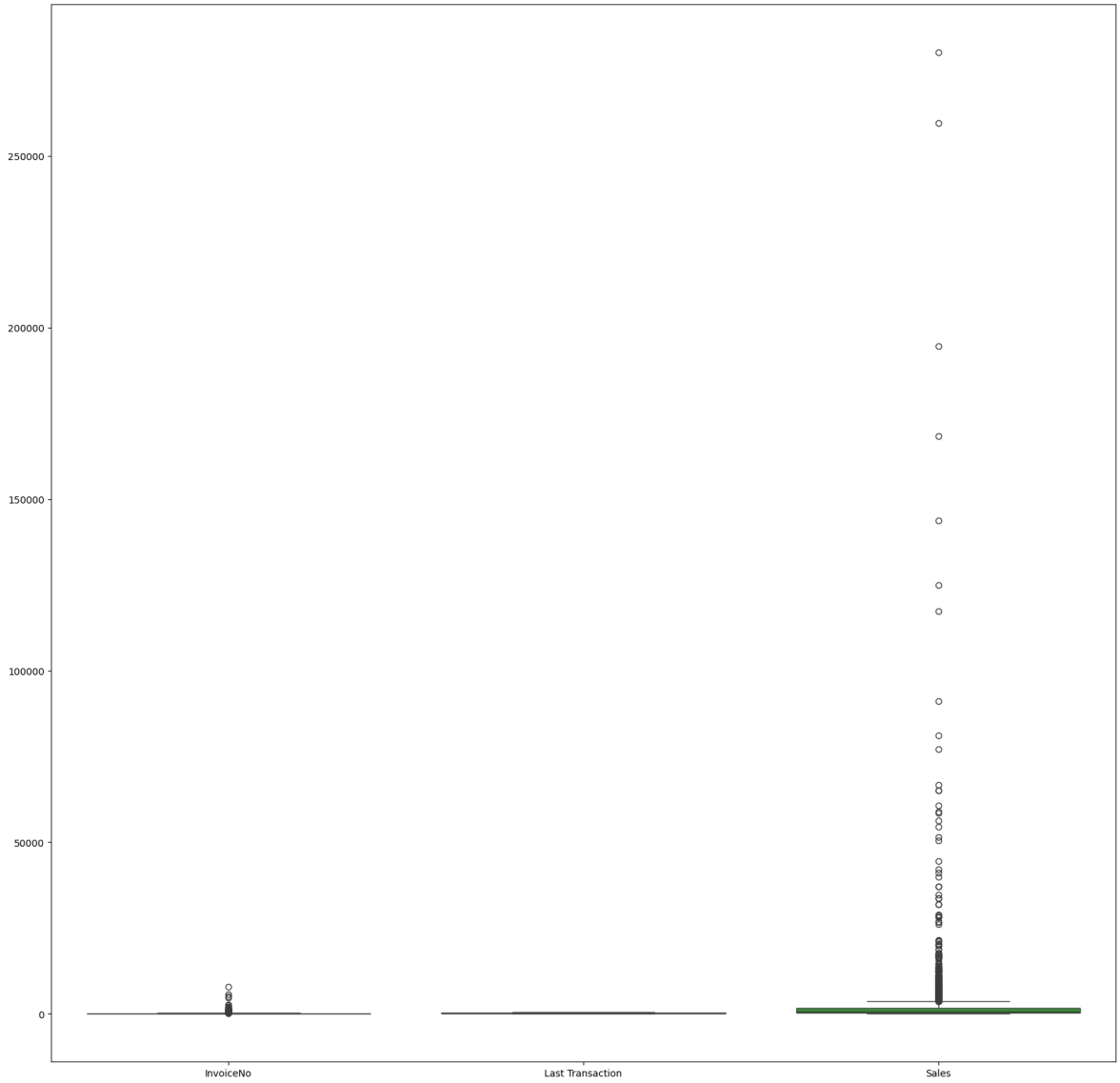
Out[27]:

	CustomerID	Country	Last Transaction	InvoiceNo	Sales
0	12346.0	United Kingdom	325	1	77183.60
1	12347.0	Iceland	366	182	4310.00
2	12348.0	Finland	357	31	1797.24
3	12349.0	Italy	18	73	1757.55
4	12350.0	Norway	309	17	334.40
...	...	...	...	...	...
4342	18280.0	United Kingdom	277	10	180.60
4343	18281.0	United Kingdom	180	7	80.82
4344	18282.0	United Kingdom	125	12	178.05
4345	18283.0	United Kingdom	336	756	2094.88
4346	18287.0	United Kingdom	201	70	1837.28

4347 rows × 5 columns

Removal of Outliers using boxplot

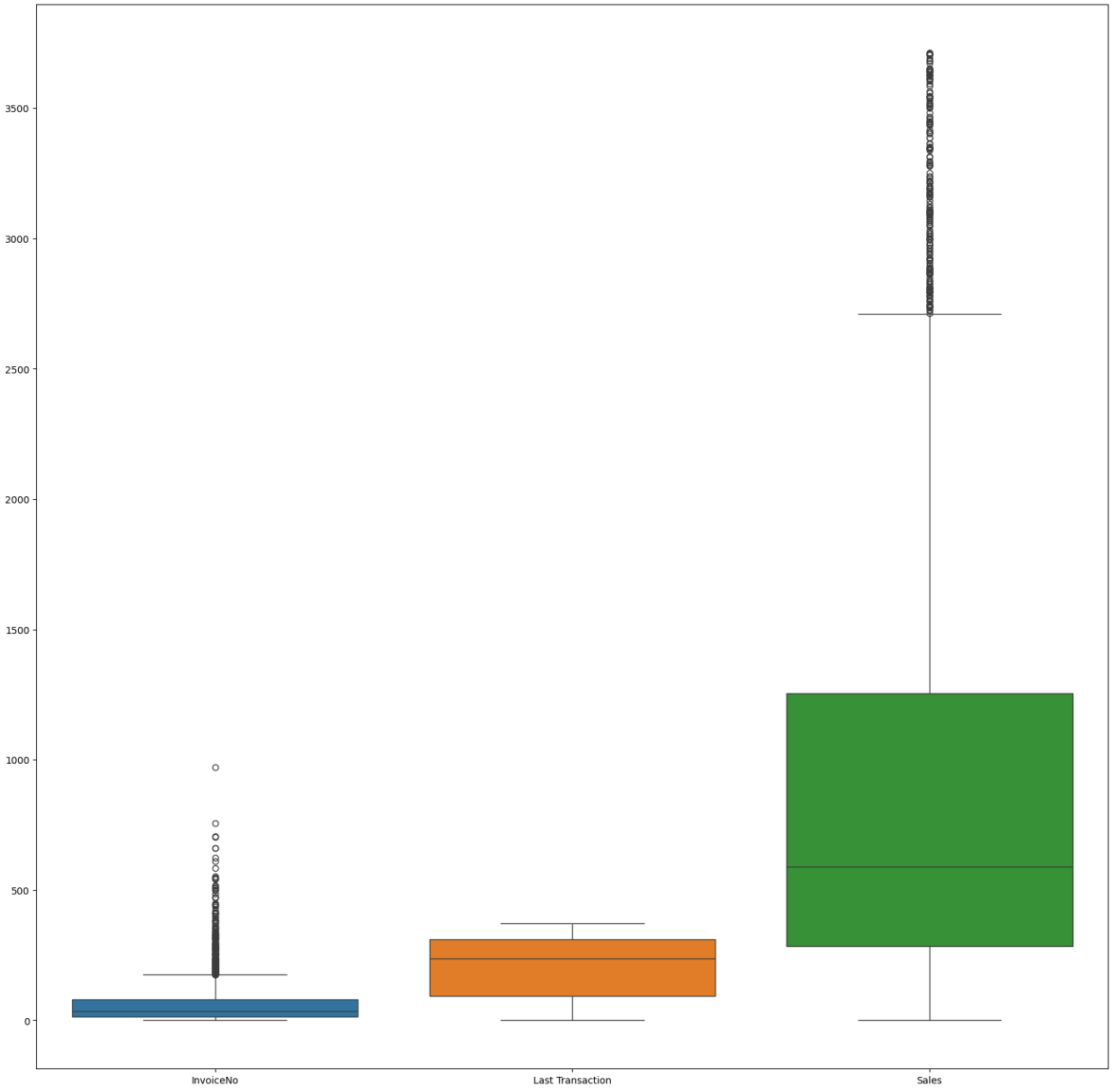
In [28]:
```
plt.figure(figsize=(20,20))
sns.boxplot(data = new_df[['InvoiceNo', 'Last Transaction', 'Sales']])
plt.show()
```



In [29]:
```
IQR = new_df['Sales'].quantile(0.75) - new_df['Sales'].quantile(0.25)
lower_limit = new_df['Sales'].quantile(0.25) - 1.5*IQR
upper_limit = new_df['Sales'].quantile(0.75) + 1.5*IQR
new_df_iqr = new_df[(new_df['Sales'] < upper_limit) & (new_df['Sales']>lo
new_df_iqr.shape
```

Out[29]: (3923, 5)

In [30]:
```
plt.figure(figsize=(20,20))
sns.boxplot(data = new_df_iqr[['InvoiceNo', 'Last Transaction', 'Sales']]
plt.show()
```



In [31]:
```
new_df_iqr.reset_index(drop=True, inplace=True)
new_df_iqr
```

Out[31]:

	CustomerID	Country	Last Transaction	InvoiceNo	Sales
0	12348.0	Finland	357	31	1797.24
1	12349.0	Italy	18	73	1757.55
2	12350.0	Norway	309	17	334.40
3	12352.0	Norway	296	85	2506.04
4	12353.0	Bahrain	203	4	89.00
...	...	...	...	...	...
3918	18280.0	United Kingdom	277	10	180.60
3919	18281.0	United Kingdom	180	7	80.82
3920	18282.0	United Kingdom	125	12	178.05
3921	18283.0	United Kingdom	336	756	2094.88
3922	18287.0	United Kingdom	201	70	1837.28

3923 rows × 5 columns

Visualization of correlation matrix using heatmap

In [ ]:
```
plt.figure(figsize=(20,20))
sns.heatmap(new_df_iqr.corr(),cmap="Greens", annot=True)
plt.show()
```

Normalization of given dataset using MinMaxScaler

In [35]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [36]:
```
new2_df= new_df_iqr[['Last Transaction','InvoiceNo','Sales']]
scaler = MinMaxScaler()
scaled_df = scaler.fit_transform(new2_df)
scaled_df = pd.DataFrame(scaled_df)
scaled_df.columns = ['Last Transaction','InvoiceNo','Sales']
scaled_df['Country'] = new_df_iqr['Country']
scaled_df
```

Out[36]:

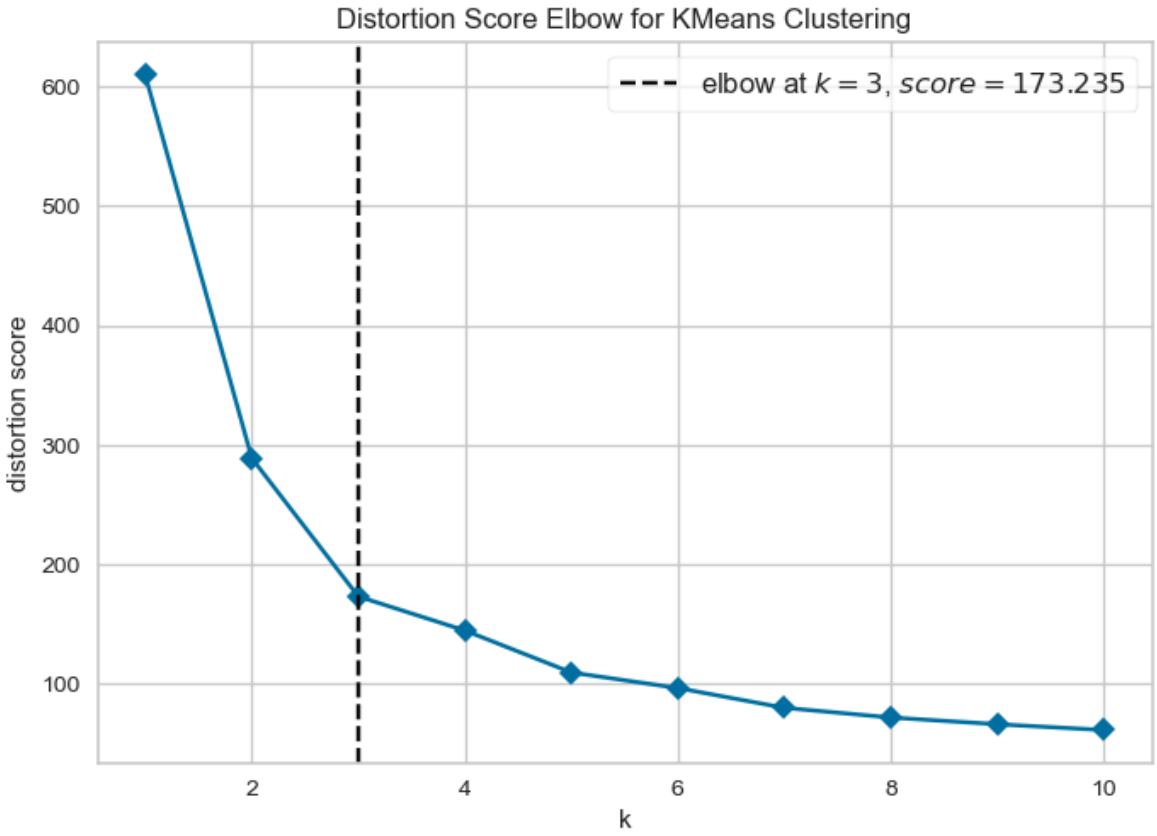
	Last Transaction	InvoiceNo	Sales	Country
0	0.957105	0.030960	0.484200	Finland
1	0.048257	0.074303	0.473507	Italy
2	0.828418	0.016512	0.090092	Norway
3	0.793566	0.086687	0.675160	Norway
4	0.544236	0.003096	0.023978	Bahrain
...	...	...	...	...
3918	0.742627	0.009288	0.048656	United Kingdom
3919	0.482574	0.006192	0.021774	United Kingdom
3920	0.335121	0.011352	0.047969	United Kingdom
3921	0.900804	0.779154	0.564388	United Kingdom
3922	0.538874	0.071207	0.494988	United Kingdom

3923 rows × 4 columns

K Means Clustering - Plot the graph using elbow method

In [ ]:
```
# !conda install -c districtdatalabs yellowbrick
```

In [39]:
```
df_k=scaled_df.drop(columns=['Country'],axis=1)
# Elbow Method for K means
# Import ElbowVisualizer
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
# k is range of number of clusters.
visualizer = KElbowVisualizer(model, k=(1,11), timings= False)
visualizer.fit(df_k)        # Fit data to visualizer
visualizer.show()
```



Out[39]: <Axes: title={'center': 'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

In [41]:
```
km = KMeans(n_clusters=3)
y_predicted = km.fit_predict(df_k)
df_k['clusters']= y_predicted
df_k
```

Out[41]:

	Last Transaction	InvoiceNo	Sales	clusters
0	0.957105	0.030960	0.484200	0
1	0.048257	0.074303	0.473507	1
2	0.828418	0.016512	0.090092	2
3	0.793566	0.086687	0.675160	0
4	0.544236	0.003096	0.023978	2
...	...	...	...	...
3918	0.742627	0.009288	0.048656	2
3919	0.482574	0.006192	0.021774	1
3920	0.335121	0.011352	0.047969	1
3921	0.900804	0.779154	0.564388	0
3922	0.538874	0.071207	0.494988	0

3923 rows × 4 columns

In [42]:
```
km.cluster_centers_
```

Out[42]:
```
array([[0.79768705, 0.15811971, 0.63364003],
       [0.2209531 , 0.04001379, 0.14854249],
       [0.7769415 , 0.04284432, 0.15758673]])
```

Model Evaluation

In [43]:
```
from sklearn.metrics import silhouette_samples, silhouette_score
score = silhouette_score(df_k, km.labels_, metric='euclidean')
print(score)
```

```
0.7825013360584594
```