Experimentation on Online Retail Data using two clustering Algorithms to discover distinct groups.

By Dayo Samuel, 2022

Abstract

The goal of this research is to use clustering analysis to identify distinct groups of customers in an online retail database, with the purpose of creating targeted marketing campaigns. To achieve this, I will apply two different clustering algorithms and compare their results in order to make the best decision. The scikitlearn implementation of K-means and DBSCAN will be used to determine the optimal number of clusters through the elbow method. Additionally, I will utilize principal component analysis (PCA) to reduce the dimensionality of the data, allowing for the visualization of the clustering results on higher dimensional datasets

Keywords: Clustering Analysis, Targeted Marketing, Principal Component Analysis

3.0 Introduction

Clustering is a data analysis technique that involves grouping data points together based on their similarity. This can be useful in many different contexts, including identifying different customer groups with similar characteristics, dividing images into segments based on pixel features, and more. For instance, a company might use clustering to group its customers into different segments based on their purchasing habits, which can help them tailor marketing campaigns and increase sales. In image segmentation, clustering algorithms can be used to divide pixels into different groups based on characteristics like colour and texture, which can be useful for tasks like object detection and image analysis

Ashishkumar et al (2014) explored ways to effectively analyse and segment large datasets, known as big data, which often have a high volume and many dimensions. They used a data reduction technique called RFM (Recency, Frequency, Monetary) analysis on a large dataset and found ways to optimize clustering techniques for segmenting the data through data partitioning and parallelization. The researchers also performed analysis on the segments of the data that resulted from the clustering experiments and found that it is worthwhile to delve deeper into the segments to gain more insights into the behaviour of businesses.

Daqing Chen et al (2012) conducted a study on retail data to help a business understand its customers and improve its customer-centric marketing efforts. Using the Recency, Frequency, and Monetary model, the business's customers were segmented into various groups using the k-means clustering algorithm and decision tree induction. The main characteristics of the consumers in each segment were then identified. Based on these findings, a set of recommendations was provided to the business on how to implement consumer-centric marketing strategies.

In 2019, Daqing et al conducted an experimental study to compare different methods for predicting customer profitability over time. Like other researchers, they used the Recency, Frequency, and Monetary (RFM) model to measure customer profitability. At one point in the study, they used k-means clustering to divide customers into high, medium, and low groups based on their RFM values. The RFM model was chosen because it is simple and easy to understand in practice.

In this work, the aim is to utilize clustering analysis to find distinct groups of customers in an online retail database, in order to create targeted marketing programs. Two different clustering algorithms will be used, and their results will be compared to make the most informed decision. The scikit-learn implementation of K-means and DBSCAN will be used to determine the optimal number of clusters through the elbow method. Additionally, principal component analysis (PCA) will be applied to decrease the dimensionality of the data, which will enable the visualization of the clustering results for higher dimensional datasets.

3.1 Datasets

The Data used for the experimentation on online retail data using clustering Algorithms gotten from the UCI (UCI Machine Learning Repository: Online Retail Data Set), which was credited to Dr Daqing Chen, Director (Public Analytics group) as the source. The dataset is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers. There are 11 variables (attributes) in the data set as shown in Table 2, and it contains all the transactions occurring in years mentioned. In this study, only 2011 transactions generated are explored. Over that period, there were 18,291 valid transactions in total, associated with some 4381 valid distinct UK postcodes.

Table 2 Variables in the customer transaction data set

InvoiceNo	Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation
StockCode	Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.
Description	Product (item) name. Nominal.
Quantity	The quantities of each product (item) per transaction. Numeric.
InvoiceDate	Invoice Date and time. Numeric, the day and time when each transaction was generated
UnitPrice	Unit price. Numeric, Product price per unit in sterling.
CustomerID	Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.
Country	Country name. Nominal, the name of the country where each customer resides.

3.2 Explanation and Preparation of dataset

Adequate pre-processing was conducted to address null values, Outliers, Negative values which are not usual and duplicate with opposite signs. The dataset used was pre – processed are:

1. Lots of nan values

2. The Quantity had negative values, which does not make sense
3. Lots of duplicate values with opposite signs around 50 of them were removed
4. Some missing data rows were discarded.
5. Since most of the data (about 96%) had been concentrated in the UnitPrice less than 100, I found better results excluding the values above it!

The data set has some important columns like the Product Name and based on the quantity sold and unit price we wish to establish some important business decisions. The dataset provided will be applied different clustering methods and results are summarized below

Dataset Preparation and Processing:

As the dataset contains records from 2 years of data 2010 and 2011. As part of the data analysis, the analysis will be on only 2011 data. With Data Processing, the key indicators are to be extracted and cleaned for the analysis. The dataset contains many null customer's ids; however, they would not play any major role as part of the product classification so I would drop the record for the same. The following steps are the processing carried out:

Extracting year from invoice date

The year 2011 data was extracted using the invoice date, the unique customers and products code checked. The dataset contains many null Customer Ids but since it will not play role as part of Product Classification so will not be dropping records for the same. After extracting data for the necessary time period, the following insights were obtained: the top countries with the largest number of customers, the month with the highest sales, and the product or stock code that contributes the most to sales. The increasing sales month over month suggest that retailers are expanding their businesses and boosting their sales. Additionally, we checked the seasonality of sales for different stock codes month over month and found that the DOT stock code product is consistently strong, indicating that it is not seasonal.

Data Pre-Processing

Label encoding is a pre-processing step that I used to handle string values in my data. A label encoder object is a tool that is able to understand and process word labels. In this case, I used a label encoder object to pre-process my data by converting the string values to numerical labels. This is useful because many machine learning algorithms are not able to directly process string values and require numerical data as input. By converting the string values to numerical labels, I was able to apply these algorithms more easily to my data. After identifying the important features for classification, I used standard scaling to scale the data. I then checked the within-cluster sum of squares (WCSS) on both the scaled and original data, as shown in Figure 1 and Figure 2. Standard scaling is a common pre-processing step that is used to transform the features of a dataset so that they have a mean of 0 and a standard deviation of 1. This is useful because many machine learning algorithms are sensitive to the scale of the input features and can perform poorly if the features have very different scales. By scaling the data, I aimed to ensure that all of the features were on a similar scale and would be weighted equally by the machine learning algorithm. The WCSS is a measure of the compactness of the clusters in a clustering algorithm and can be used to determine the optimal number of clusters. By comparing the WCSS on the scaled and original data, I was able to see the impact of scaling on the clustering results.

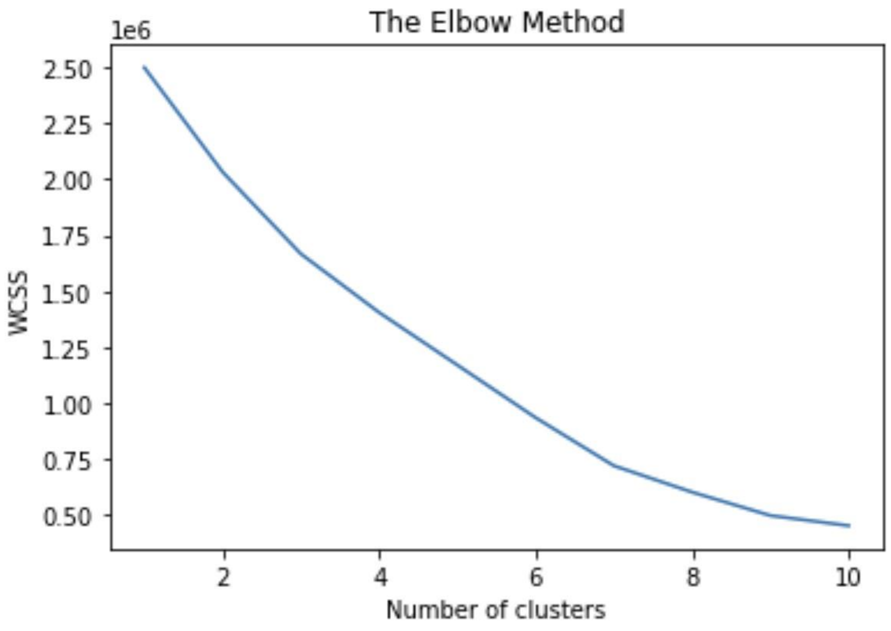


Figure 1 WCSS on Scaled Data

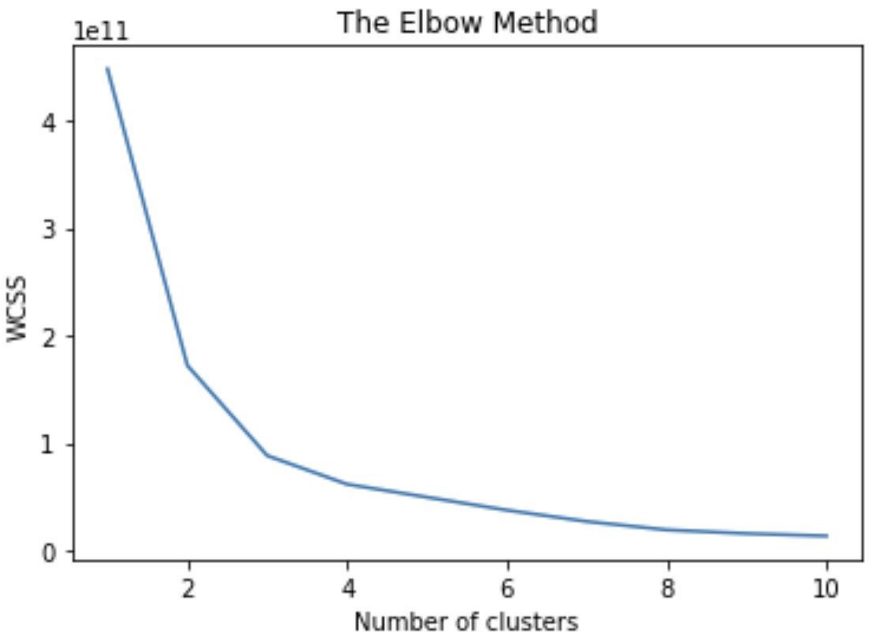


Figure 2 WCSS on original data

3.3 Clustering Methodology

K-Means Clustering algorithm

K-means clustering is a popular clustering algorithm that groups data points into a specified number of clusters (k) based on their similarity. The algorithm works by first initializing k centroids, which are points representing the centre of each cluster. The data points are then assigned to the cluster corresponding to the nearest centroid. Next, the centroids are updated to the mean of the points in the corresponding cluster. This process is repeated until the centroids no longer move or the desired level of convergence is reached. The goal of the algorithm is to minimize the distance between the points in a cluster and the corresponding centroid. K-means is a simple and effective algorithm that is widely used in a variety of applications.

To calculate the K-means clustering algorithm, we need to follow the below steps:

- Initialize k centroids (randomly or using some heuristic)
- Assign each data point to the closest centroid
- Update the centroids to the mean of the points in the corresponding cluster
- Repeat steps 2 and 3 until the centroids no longer move or the desired level of convergence is reached

To calculate the distance between a data point and a centroid, we can use the Euclidean distance, which is defined as the square root of the sum of the squared differences between the coordinates of the two points. For example, if we have a data point with coordinates (x1, y1) and a centroid with coordinates (x2, y2), the Euclidean distance between the two points would be calculated as follows:

distance = sqrt((x1-x2)^2 + (y1-y2)^2)

Once we have calculated the distance between the data point and each centroid, we can assign the data point to the cluster corresponding to the nearest centroid. After all of the data points have been assigned to clusters, we can update the centroids by taking the mean of the points in each cluster. This process is then repeated until the centroids no longer move or the desired level of convergence is reached. The steps of the K-Means Clustering algorithm done on the dataset are shown below:

Experimentation using K-means

After initializing the cluster centroids, the memberships of each cluster in each of the 13 subsets are calculated in parallel. This means that each cluster has members in each of the 13 subsets. The new centroid of each cluster is then calculated based on all of the cluster members in all of the 13 subsets. This parallel version of the K-means algorithm produces the same final set of clusters as the sequential version, but with less processing time. Once the final clusters have been created, I will check the total records and unique products in each cluster by combining the cluster results with the original data. The final number of clusters can be visualized using seaborn count plots and scatterplots, as demonstrated in figures 3 and 4.

DBSCAN Clustering algorithm

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups data points into clusters based on their density. The algorithm works by first identifying "core points" that have a high density of neighbouring points. These core points are then used to form clusters. Points that are not part of any cluster are considered "noise" and are not included in the final result.

DBSCAN has several advantages over other clustering algorithms, such as k-means. First, it does not require the user to specify the number of clusters in advance. This is because the algorithm automatically determines the number of clusters based on the density of the data. Second, DBSCAN is able to handle data with varying densities and can identify clusters of arbitrary shape.

To use the DBSCAN algorithm, the user must specify two parameters: Eps and MinPts. Eps is the maximum distance that defines a neighbourhood around a data point. MinPts is the minimum number of points that must be in a neighbourhood for a point to be considered a core point. The algorithm then uses these parameters to identify clusters and noise in the data.

Experimentation using DBSCAN

The DBSCAN algorithm is typically calculated using the following steps:

- Initialize an empty list of clusters and an empty list of noise points.
- For each data point, find all points within a distance Eps of the point (using a distance measure such as Euclidean distance).
- If the number of points within Eps of the point is greater than or equal to MinPts, the point is a core point. Add it to a cluster and expand the cluster to include all points within Eps of the core point.
- If the number of points within Eps of the point is less than MinPts, the point is a noise point. Add it to the list of noise points.
- Repeat this process for each data point. When all points have been processed, the algorithm will have identified all clusters and noise points in the data.

The DBSCAN algorithm can be made more efficient by using techniques such as indexing. In this case, the values of eps and min_samples were set to 0.3 and 10, respectively. These values determine the behaviour of the algorithm by specifying the maximum distance between two samples for them to be considered part of the same cluster (eps=0.3) and the minimum number of samples in a neighbourhood for a point to be considered a core point (min_samples=10). The values of eps and min_samples will affect the shape, size, and number of clusters identified by DBSCAN.

3.4 Results analysis and discussion

In this study, two algorithms were tested on large datasets, commonly referred to as "big data." The percentage of null customer IDs in the dataset was 23%, and the original data tended to produce clearer results on a WCCS plot compared to scaled data. However, the size of the data made it difficult to plot a dendrogram. Despite these challenges, the algorithms were able to effectively process and analyse the data.

As a result of the analysis, three clusters were identified in the dataset. The first cluster contained 1534 samples, the second cluster contained 1230 samples, and the third cluster contained 1230 samples. These clusters were created by dividing the samples into groups based on certain characteristics or features. The samples within each cluster were more similar to each other than they were to samples in other clusters, indicating the presence of distinct patterns or groups in the dataset. This suggests that the algorithms were successful in identifying meaningful patterns and relationships within the data.

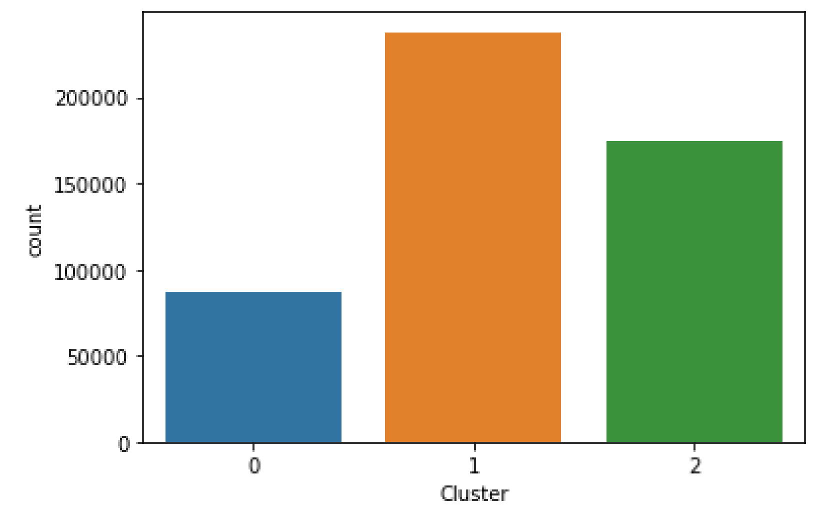


Figure 3 Clusters founded with Count plot

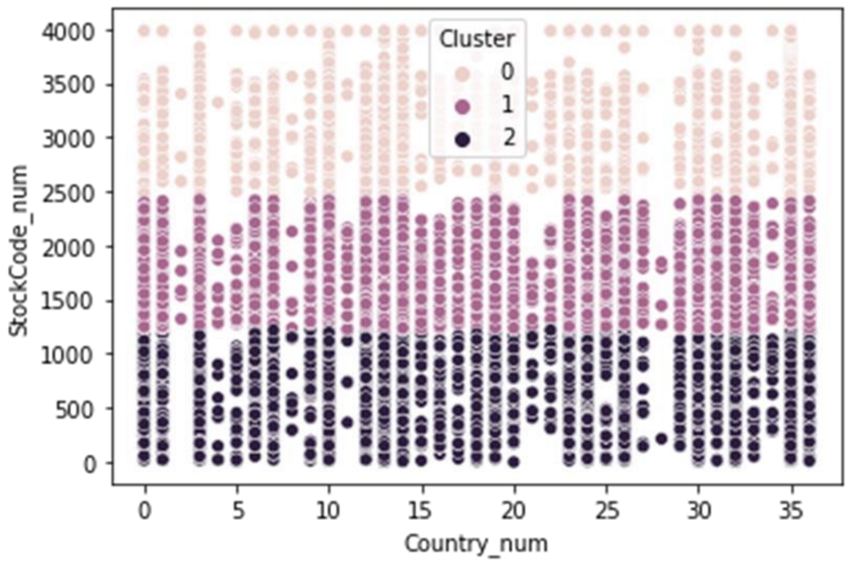


Figure 4 Clusters founded with scatter plot

Conclusions

This study aimed to analyse and identify distinct groups within a dataset. The results showed that the samples in each cluster are more similar to each other than they are to samples in other clusters. This suggests that there are distinct patterns or groups present in the dataset. In future work, it may be useful to further classify the customers in the cluster with the highest number in order to identify the high-quality customers within that group. Overall, the clustering analysis performed in this study provided insight into the relationships and patterns present in the dataset.

References

Ashishkumar Singh, Grace Rumantir, Annie South, and Blair Bethwaite. 2014. Clustering Experiments on Big Transaction Data for Market Segmentation. In Proceedings of the 2014 International Conference on Big Data Science and Computing (BigDataScience '14). *Association for Computing Machinery, New York*, NY, USA, Article 16, 1–7. https://doi.org/10.1145/2640087.2644161

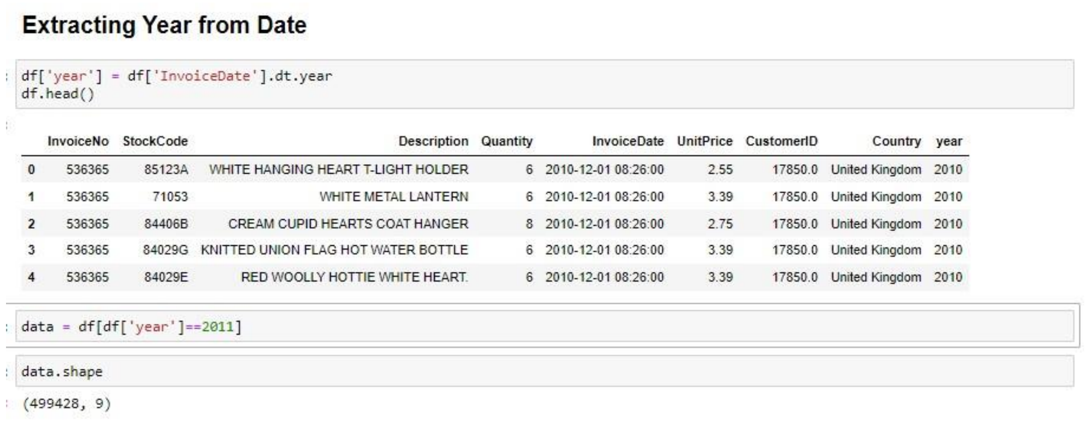
Chen, Daqing & Guo, Kun & Li, Bo. (2019). Predicting Customer Profitability Dynamically over Time: An Experimental Comparative Study.

Appendix One

(These screenshots depict the referenced procedures that were completed for Task 3. The source of the information is a combination of self-written notes and a Jupyter notebook.)

1. Dataset Preparation and Processing:

- Extracting year from invoice date



Extracting Year from Date

```
df['year'] = df['InvoiceDate'].dt.year
df.head()
```

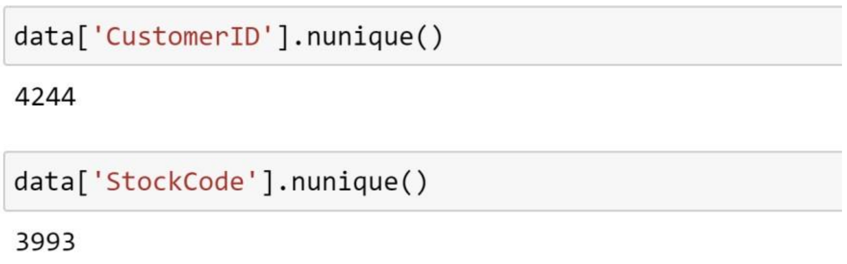
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom	2010
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom	2010
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom	2010

```
data = df[df['year']==2011]
```

```
data.shape
```

```
(499428, 9)
```

- Unique Customers and Products/Stock Code



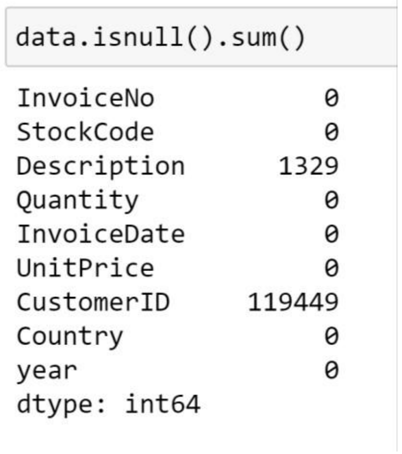
```
data['CustomerID'].nunique()
```

```
4244
```

```
data['StockCode'].nunique()
```

```
3993
```

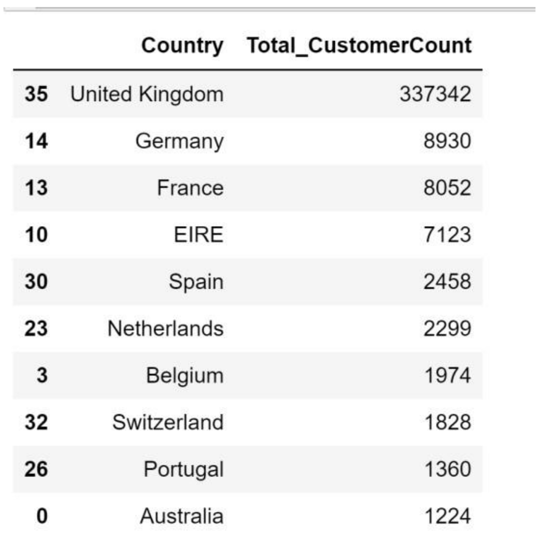
- Check for Null values in the dataset



```
data.isnull().sum()
```

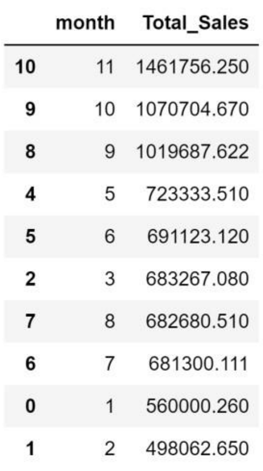
```
InvoiceNo           0
StockCode           0
Description      1329
Quantity            0
InvoiceDate         0
UnitPrice           0
CustomerID     119449
Country             0
year                0
dtype: int64
```

- Top Countries with the greatest number of customers



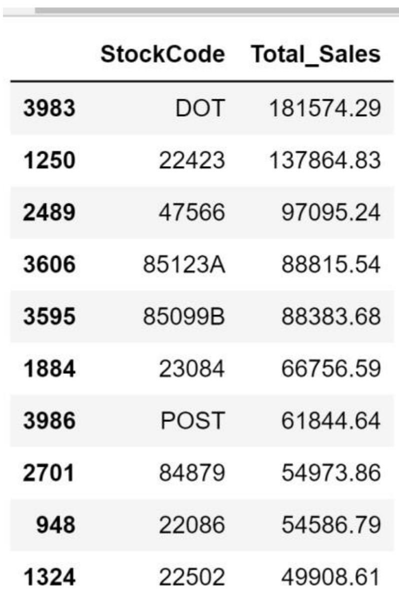
	Country	Total_CustomerCount
35	United Kingdom	337342
14	Germany	8930
13	France	8052
10	EIRE	7123
30	Spain	2458
23	Netherlands	2299
3	Belgium	1974
32	Switzerland	1828
26	Portugal	1360
0	Australia	1224

- Month having highest sales



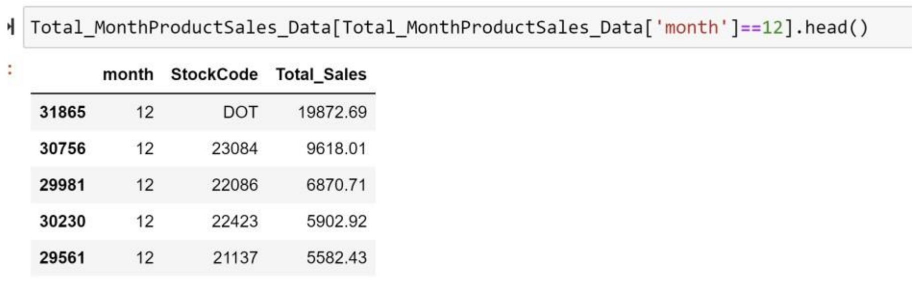
	month	Total_Sales
10	11	1461756.250
9	10	1070704.670
8	9	1019687.622
4	5	723333.510
5	6	691123.120
2	3	683267.080
7	8	682680.510
6	7	681300.111
0	1	560000.260
1	2	498062.650

- Which Product or Stock Code contributes to most of the sales



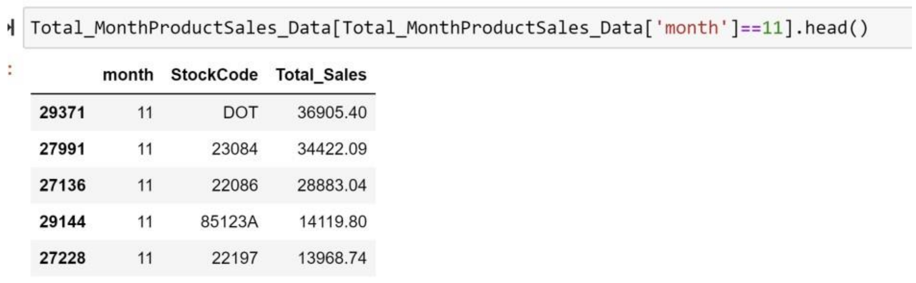
	StockCode	Total_Sales
3983	DOT	181574.29
1250	22423	137864.83
2489	47566	97095.24
3606	85123A	88815.54
3595	85099B	88383.68
1884	23084	66756.59
3986	POST	61844.64
2701	84879	54973.86
948	22086	54586.79
1324	22502	49908.61

- Check for seasonality in sales for different Stock Codes month over mo



```
Total_MonthProductSales_Data[Total_MonthProductSales_Data['month']==12].head()
```

	month	StockCode	Total_Sales
31865	12	DOT	19872.69
30756	12	23084	9618.01
29981	12	22086	6870.71
30230	12	22423	5902.92
29561	12	21137	5582.43

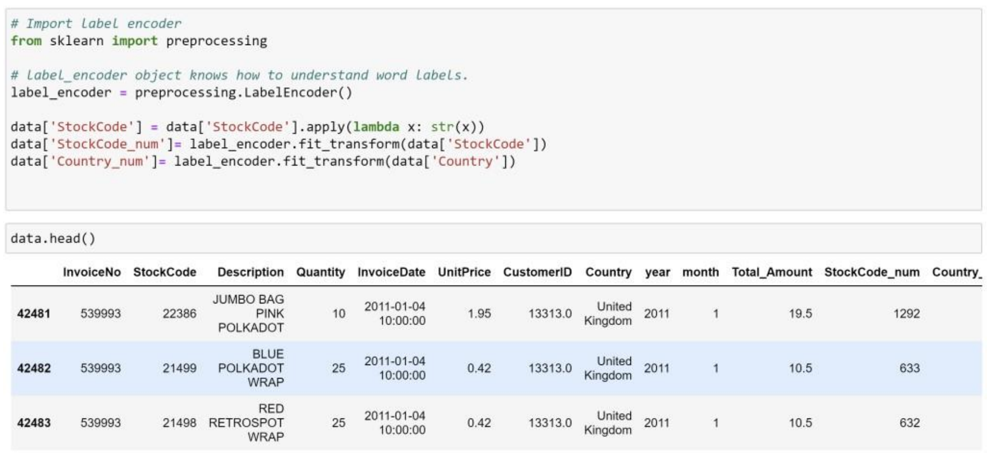


```
Total_MonthProductSales_Data[Total_MonthProductSales_Data['month']==11].head()
```

	month	StockCode	Total_Sales
29371	11	DOT	36905.40
27991	11	23084	34422.09
27136	11	22086	28883.04
29144	11	85123A	14119.80
27228	11	22197	13968.74

2. Data Pre-Processing

- Label Encoding Data to handle String values

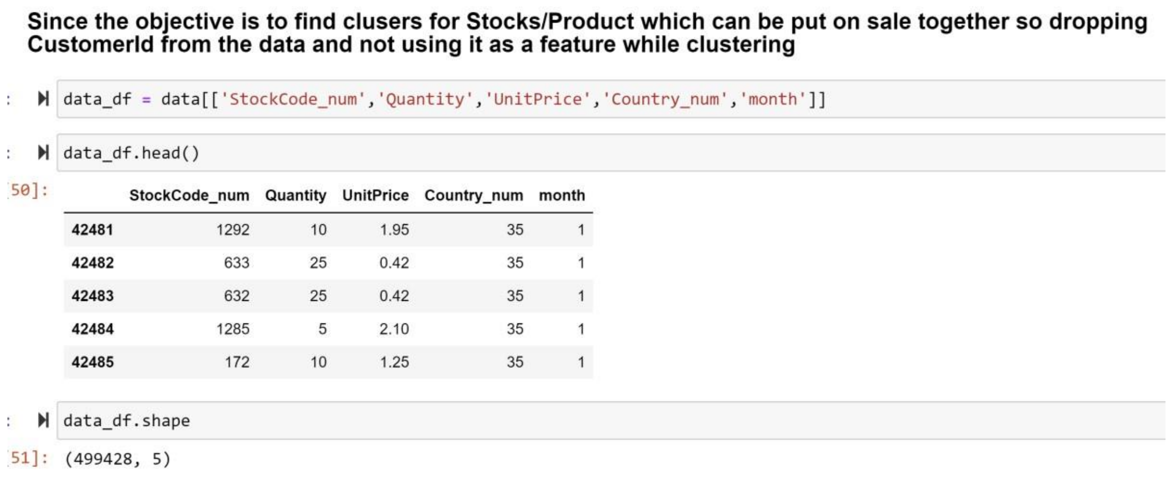


```
# Import label encoder
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

data['StockCode'] = data['StockCode'].apply(lambda x: str(x))
data['StockCode_num']= label_encoder.fit_transform(data['StockCode'])
data['Country_num']= label_encoder.fit_transform(data['Country'])
```

```
data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year	month	Total_Amount	StockCode_num	Country_
42481	539993	22386	JUMBO BAG PINK POLKADOT	10	2011-01-04 10:00:00	1.95	13313.0	United Kingdom	2011	1	19.5	1292	
42482	539993	21499	BLUE POLKADOT WRAP	25	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011	1	10.5	633	
42483	539993	21498	RED RETROSPOT WRAP	25	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011	1	10.5	632	

- Extracting important features for classification



Since the objective is to find clusers for Stocks/Product which can be put on sale together so dropping CustomerId from the data and not using it as a feature while clustering

```python
data_df = data[['StockCode_num','Quantity','UnitPrice','Country_num','month']]
```

```python
data_df.head()
```

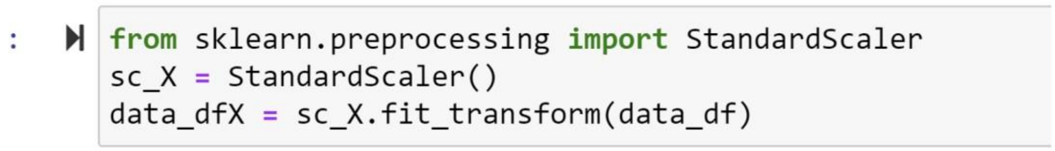
50]:

	StockCode_num	Quantity	UnitPrice	Country_num	month
42481	1292	10	1.95	35	1
42482	633	25	0.42	35	1
42483	632	25	0.42	35	1
42484	1285	5	2.10	35	1
42485	172	10	1.25	35	1

```python
data_df.shape
```

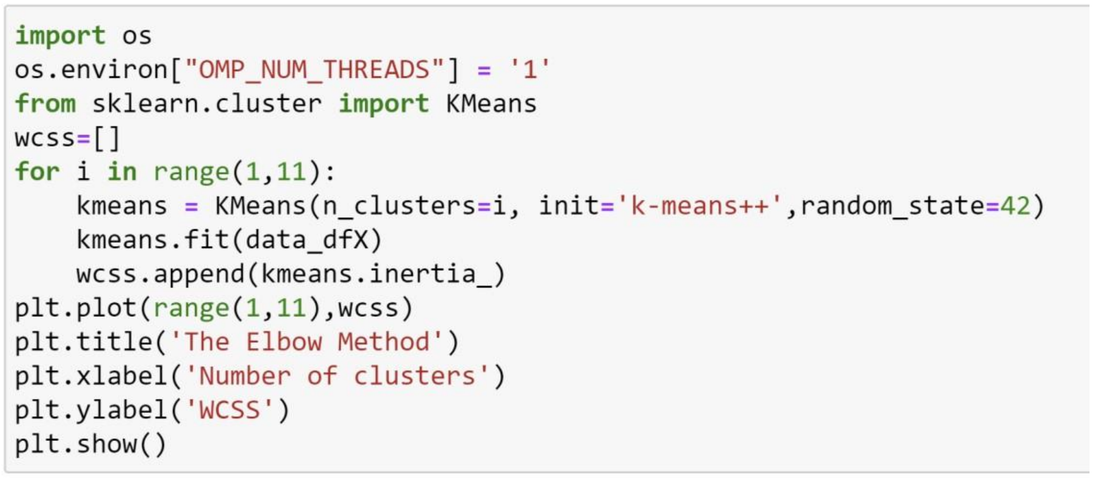
```
51]: (499428, 5)
```

- Scaling Data



```python
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
data_dfX = sc_X.fit_transform(data_df)
```

- Checking WCSS on Scaled Data



```python
import os
os.environ["OMP_NUM_THREADS"] = '1'
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state=42)
    kmeans.fit(data_dfX)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

- Checking WCSS on original data

```
import os
os.environ["OMP_NUM_THREADS"] = '1'
from sklearn.cluster import KMeans
wcss=[]
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state=42)
    kmeans.fit(data_df)
    wcss.append(kmeans.inertia_)
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

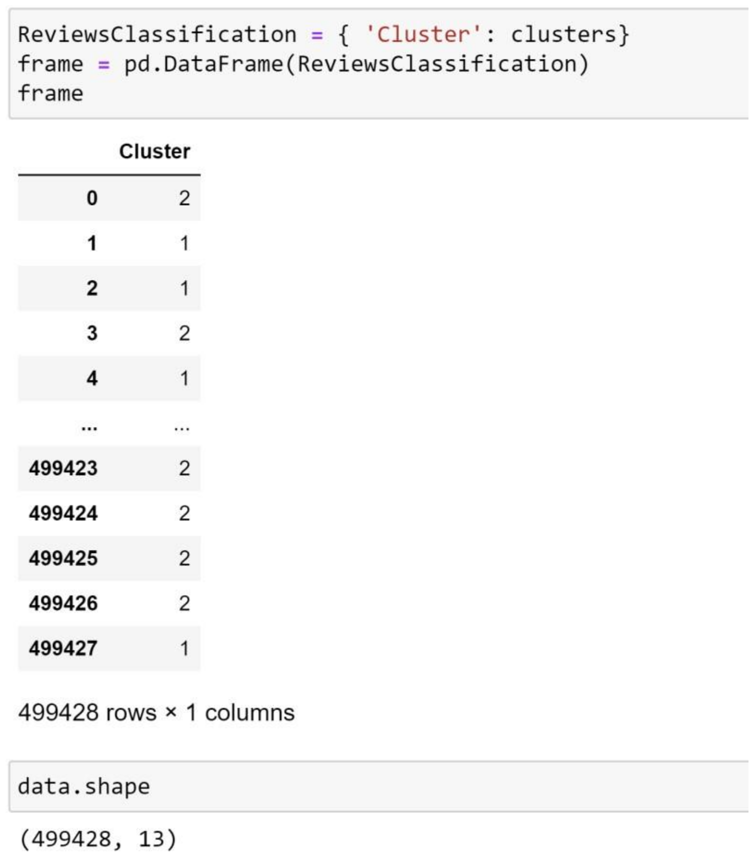
3. Experimentation using K-means

- Applying Kmeans Classification

```
# implement kmeans
num_clusters = 3
km = KMeans(n_clusters=num_clusters)
km.fit(data_df)
clusters = km.labels_.tolist()
```

```
label = km.predict(data_df)
```

• Check on the final Clusters created



```
ReviewsClassification = { 'Cluster': clusters}
frame = pd.DataFrame(ReviewsClassification)
frame
```

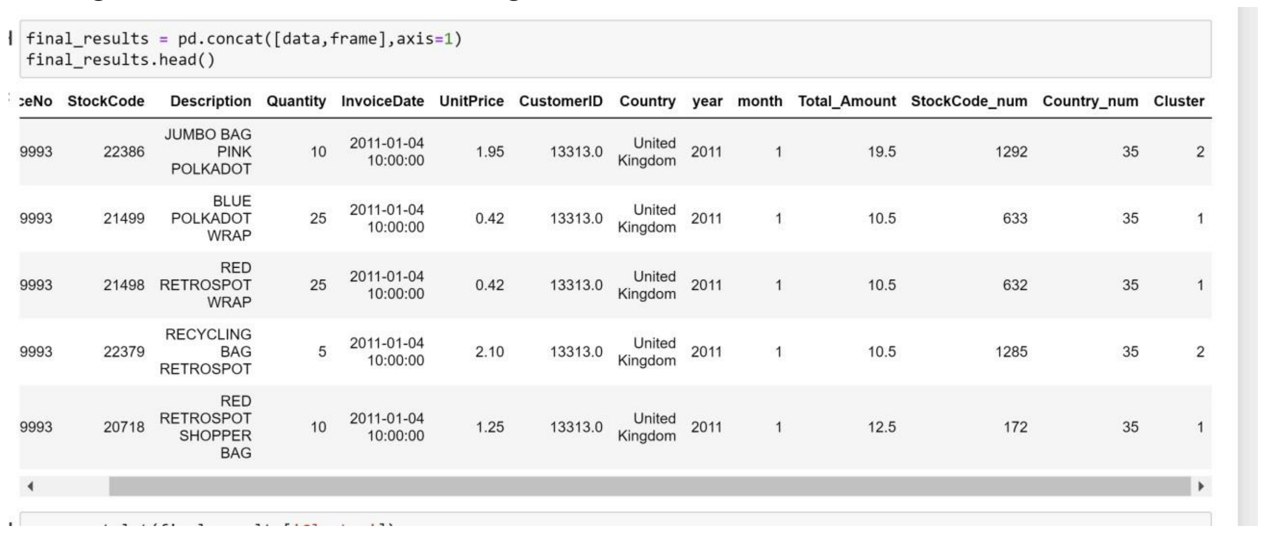
	Cluster
0	2
1	1
2	1
3	2
4	1
...	...
499423	2
499424	2
499425	2
499426	2
499427	1

499428 rows × 1 columns

```
data.shape
```

```
(499428, 13)
```

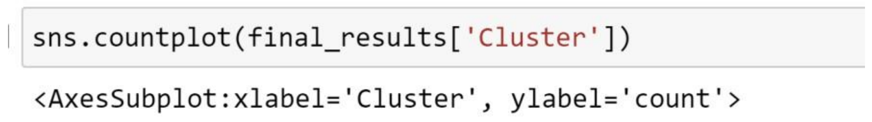
• Combining cluster results formed with original data



```
final_results = pd.concat([data,frame],axis=1)
final_results.head()
```

ceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	year	month	Total_Amount	StockCode_num	Country_num	Cluster
9993	22386	JUMBO BAG PINK POLKADOT	10	2011-01-04 10:00:00	1.95	13313.0	United Kingdom	2011	1	19.5	1292	35	2
9993	21499	BLUE POLKADOT WRAP	25	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011	1	10.5	633	35	1
9993	21498	RED RETROSPOT WRAP	25	2011-01-04 10:00:00	0.42	13313.0	United Kingdom	2011	1	10.5	632	35	1
9993	22379	RECYCLING BAG RETROSPOT	5	2011-01-04 10:00:00	2.10	13313.0	United Kingdom	2011	1	10.5	1285	35	2
9993	20718	RED RETROSPOT SHOPPER BAG	10	2011-01-04 10:00:00	1.25	13313.0	United Kingdom	2011	1	12.5	172	35	1

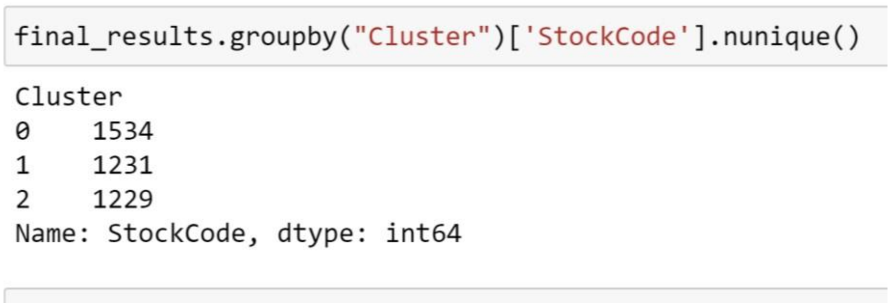
• Total Records in each cluster



```
sns.countplot(final_results['Cluster'])
```

```
<AxesSubplot:xlabel='Cluster', ylabel='count'>
```

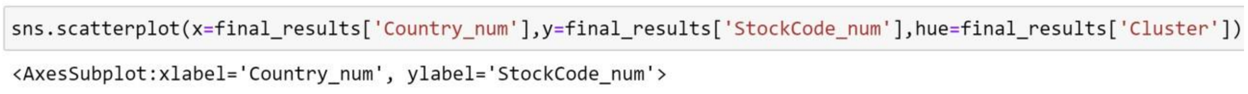
• Total Unique Product in Each Cluster



```
final_results.groupby("Cluster")['StockCode'].nunique()
```

```
Cluster
0    1534
1    1231
2    1229
Name: StockCode, dtype: int64
```

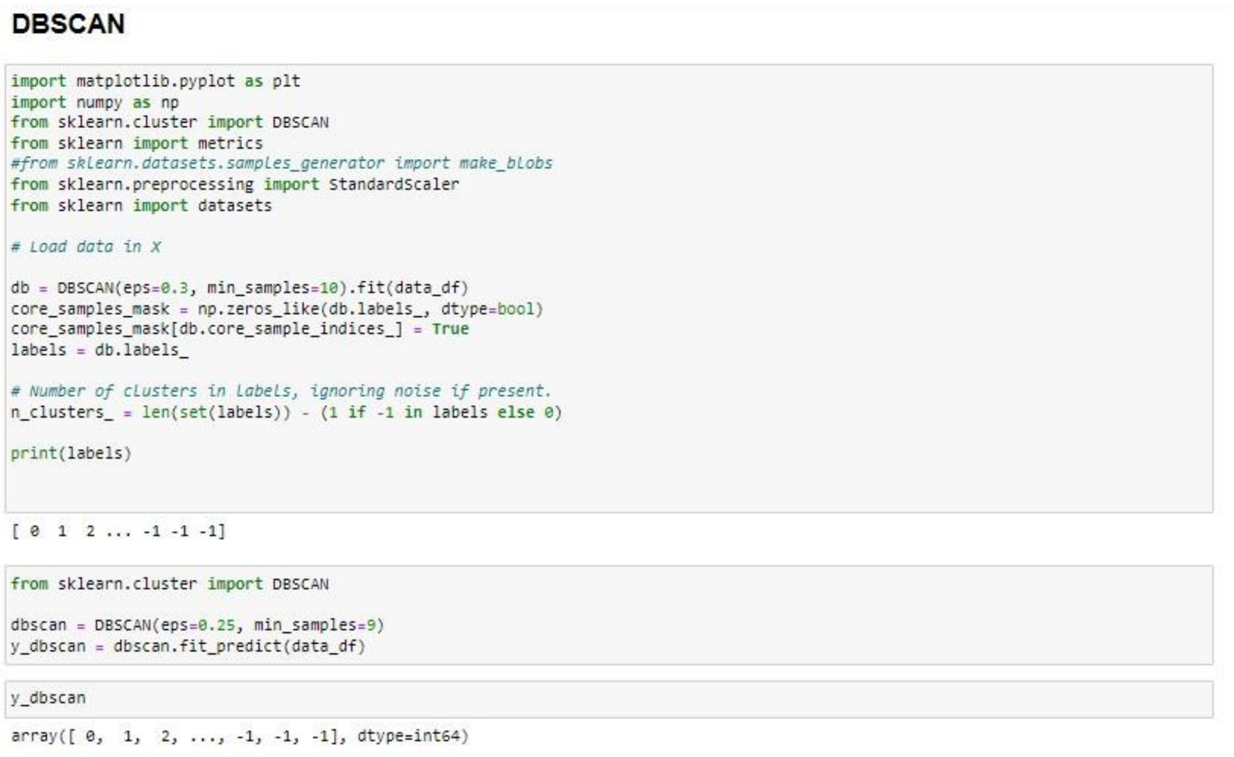
• Cluster Visualization



```
sns.scatterplot(x=final_results['Country_num'],y=final_results['StockCode_num'],hue=final_results['Cluster'])
```

```
<AxesSubplot:xlabel='Country_num', ylabel='StockCode_num'>
```

4. Experimentation using DBSCAN



DBSCAN

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
#from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

# Load data in X

db = DBSCAN(eps=0.3, min_samples=10).fit(data_df)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

print(labels)
```

```
[ 0  1  2 ... -1 -1 -1]
```

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.25, min_samples=9)
y_dbscan = dbscan.fit_predict(data_df)
```

```
y_dbscan
```

```
array([ 0,  1,  2, ..., -1, -1, -1], dtype=int64)
```