CUSTOMER SEGMENTATION USING DATA SCIENCE

Customer segmentation models are often used for dividing a company’s clients into different user groups. Customers in each group display shared characteristics that distinguish them from other users.

Here is a simple example of how companies use data segmentation to drive sales:

Every time I visit an e-commerce site, I look for items that are on sale to add to my cart. If I want to buy an item of clothing and it isn’t currently on sale, I wait until I see a special offer before making a purchase.

Data scientists at e-commerce companies often build customer segmentation models to identify shared traits amongst their customers. After building such a model, they notice that there are a handful of customers like me who always wait for a special offer before making purchases.

They classify us into a segment called “thrifty shoppers.”

Every time a new promotion is released, the company’s marketing team sends me and every other “thrifty shopper” a curated advertisement highlighting product affordability.

Whenever I get notified of a special discount, I rush to purchase all the items I require before the promotion ends, which increases the company’s sales.

Similarly, all the platform’s customers are grouped into different segments and sent targeted promotions based on their purchase behavior.

The example above demonstrates how customer segmentation models add value to organizations.

Data scientists usually build customer segmentation models using unsupervised machine learning algorithms such as [K-Means clustering](https://365datascience.com/tutorials/python-tutorials/k-means-clustering/) or hierarchical clustering. These models can pick up on similarities between user groups that often go unnoticed by the human eye.

In this article, I will show you how to build a data segmentation model in Python. You will learn to prepare data for customer segmentation and to build a K-Means algorithm from scratch. We will also look at how RFM is used in marketing to analyze customer value and explore other metrics for evaluating the performance of a clustering algorithm. Finally, we’ll answer the question of how to visualize and interpret clusters for customer segmentation.

Step 1: Prerequisites for Building a Customer Segmentation Model

In this tutorial, we will be using an [E-Commerce Dataset](https://www.kaggle.com/datasets/carrie1/ecommerce-data) from Kaggle that contains transaction information from around 4,000 customers.

You need to have a Python IDE installed on your device before you can follow along with this tutorial. I suggest using a [Jupyter Notebook](https://365datascience.com/tutorials/python-tutorials/jupyter-notebook-tutorial/) to easily run the code provided and display visualizations at each step.

Also, make sure to have the following libraries installed — Numpy, Pandas, Matplotlib, Seaborn, Scikit-Learn, Kneed, and Scipy.

Step 2: Understand the Segmentation Data

Before starting any data science project, it is vital to explore the dataset and understand each variable.

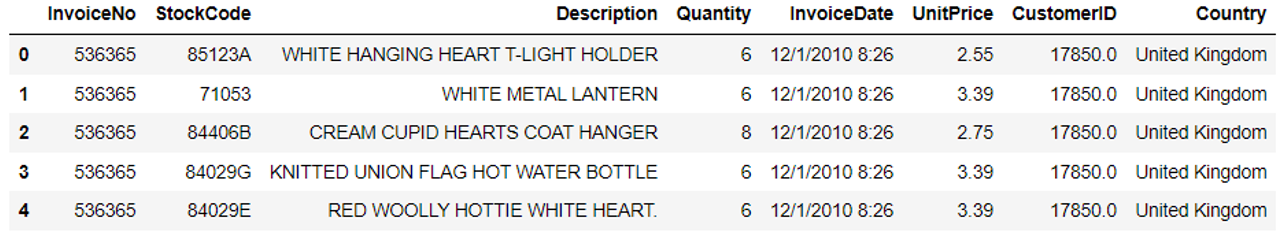
To do this, let’s import the Pandas library and load the dataset into Python:

**import** pandas **as** pd

df = pd.read\_csv('data.csv',encoding='unicode\_escape')

Now, let’s look at the head of the dataframe:

df.head()



The dataframe consists of 8 variables:

1. InvoiceNo: The unique identifier of each customer invoice.
2. StockCode: The unique identifier of each item in stock.
3. Description: The item purchased by the customer.
4. Quantity: The number of each item purchased by a customer in a single invoice.
5. InvoiceDate: The purchase date.
6. UnitPrice: Price of one unit of each item.
7. CustomerID: Unique identifier assigned to each user.
8. Country: The country from where the purchase was made.

With the transaction data above, we need to build different customer segments based on each user’s purchase behavior.

Step 3: Preprocessing Data for Segmentation

The raw data we downloaded is complex and in a format that cannot be easily ingested by customer segmentation models. We need to do some preliminary [data preparation](https://365datascience.com/trending/techniques-for-processing-traditional-and-big-data/) to make this data interpretable.

The informative features in this dataset that tell us about customer buying behavior include “Quantity”, “InvoiceDate” and “UnitPrice.” Using these variables, we are going to derive a customer’s RFM profile - Recency, Frequency, Monetary Value.

[RFM](https://ranksey.com/rfm-analysis-using-rfm-marketing-and-segmentation/) is commonly used in marketing to evaluate a client’s value based on their:

1. Recency: How recently have they made a purchase?
2. Frequency: How often have they bought something?
3. Monetary Value: How much money do they spend on average when making purchases?

With the variables in this e-commerce transaction dataset, we will calculate each customer’s recency, frequency, and monetary value. These RFM values will then be used to build the segmentation model.

**Recency**

Let’s start by calculating recency. To identify a customer’s recency, we need to pinpoint when each user was last seen making a purchase:

*# convert date column to datetime format*

df['Date']= pd.to\_datetime(df['InvoiceDate'])

*# keep only the most recent date of purchase*

df['rank'] = df.sort\_values(['CustomerID','Date']).groupby(['CustomerID'])['Date'].rank(method='min').astype(int)

df\_rec = df[df['rank']==1]

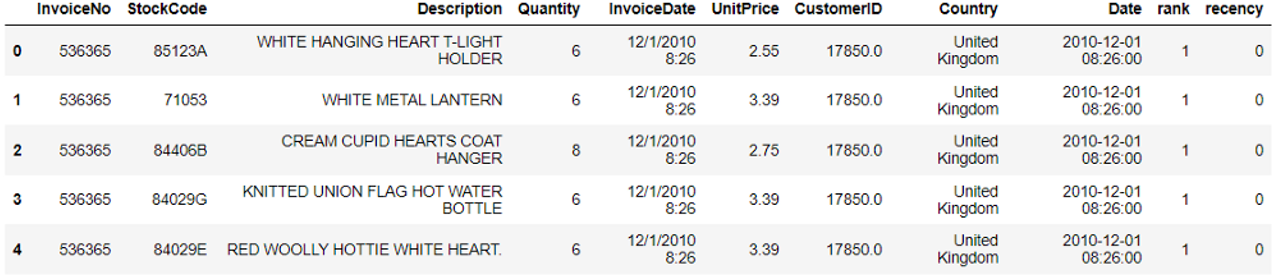
In the dataframe we just created, we only kept rows with the most recent date for each customer. We now need to rank every customer based on what time they last bought something and assign a recency score to them.

For example, if customer A was last seen acquiring an item 2 months ago and customer B did the same 2 days ago, customer B must be assigned a higher recency score.

To assign a recency score to each customerID, run the following lines of code:

df\_rec['recency'] = (df\_rec['Date'] - pd.to\_datetime(min(df\_rec['Date']))).dt.days

The dataframe now has a new column called “recency” that tells us when each customer last bought something from the platform:



**Frequency**

Now, let’s calculate frequency — how many times has each customer made a purchase on the platform:

freq = df\_rec.groupby('CustomerID')['Date'].count()

df\_freq = pd.DataFrame(freq).reset\_index()

df\_freq.columns = ['CustomerID','frequency']

The new dataframe we created consists of two columns — “CustomerID” and “frequency.” Let’s merge this dataframe with the previous one:

rec\_freq = df\_freq.merge(df\_rec,on='CustomerID')

Check the head of the dataframe to ensure that the variable “frequency” has been included:



**Monetary Value**

Finally, we can calculate each user’s monetary value to understand the total amount they have spent on the platform.

To achieve this, run the following lines of code:

rec\_freq['total'] = rec\_freq['Quantity']\*df['UnitPrice']

m = rec\_freq.groupby('CustomerID')['total'].sum()

m = pd.DataFrame(m).reset\_index()

m.columns = ['CustomerID','monetary\_value']

The new dataframe we created consists of each CustomerID and its associated monetary value. Let’s merge this with the main dataframe:

rfm = m.merge(rec\_freq,on='CustomerID')

Now, let’s select only the columns required to build the customer segmentation model:

finaldf = rfm[['CustomerID','recency','frequency','monetary\_value']]

**Removing Outliers**

We have successfully derived three meaningful variables from the raw, uninterpretable transaction data we started out with.

Before building the customer segmentation model, we first need to check the dataframe for outliers and remove them.

To get a visual representation of outliers in the dataframe, let’s create a boxplot of each variable:

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

list1 = ['recency','frequency','monetary\_value']

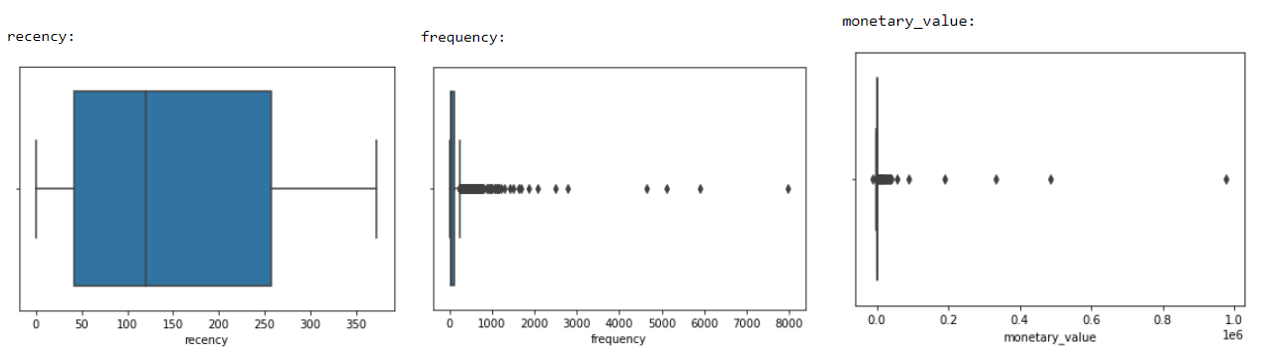
**for** i **in** list1:

print(str(i)+': ')

ax = sns.boxplot(x=finaldf[str(i)])

plt.show()

The lines of code above will generate boxplots like this for all 3 variables:



Observe that “recency” is the only variable with no visible outliers. “Frequency” and “monetary\_value”, on the other hand, have many outliers that must be removed before we proceed to build the model.

To identify outliers, we will compute a measurement called a [Z-Score](https://365datascience.com/tutorials/statistics-tutorials/significance-level-reject-region/). Z-Scores tell us how far away from the mean a data point is. A Z-Score of 3, for instance, means that a value is 3 standard deviations away from the variable’s mean.

Run the following lines of code to remove outliers in every column of our dataframe (We are going to remove every data point with a Z-Score>=3):

**from** scipy **import** stats

**import** numpy **as** np

*# remove the customer id column*

new\_df = finaldf[['recency','frequency','monetary\_value']]

*# remove outliers*

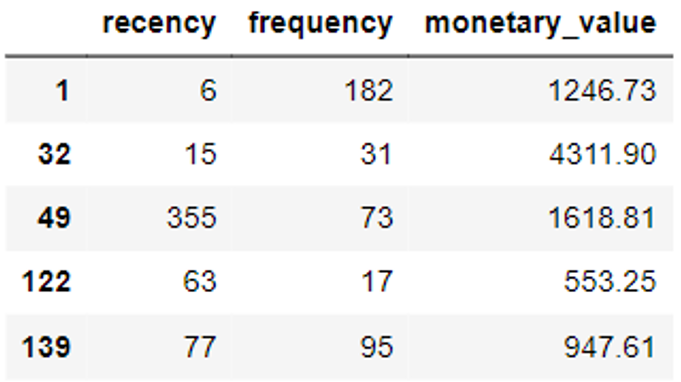
z\_scores = stats.zscore(new\_df)

abs\_z\_scores = np.abs(z\_scores)

filtered\_entries = (abs\_z\_scores < 3).all(axis=1)

new\_df = new\_df[filtered\_entries]

Looking at the head of the dataframe again, we notice that a few extreme values have been removed:



Standardization

The final pre-processing technique we will apply to the dataset is standardization.

Run the following lines of code to scale the dataset’s values so that they follow a normal distribution:

**from** sklearn.preprocessing **import** StandardScaler

new\_df = new\_df.drop\_duplicates()

col\_names = ['recency', 'frequency', 'monetary\_value']

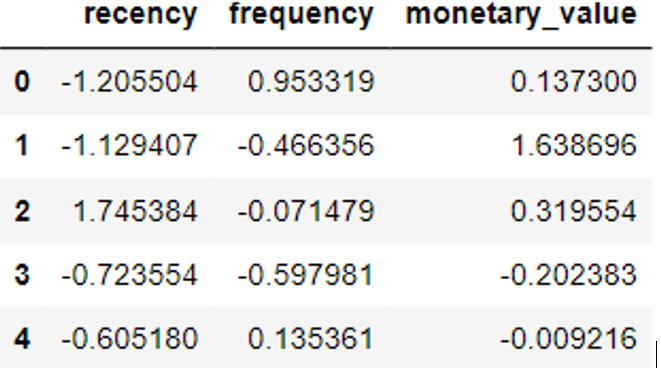
features = new\_df[col\_names]

scaler = StandardScaler().fit(features.values)

features = scaler.transform(features.values)

scaled\_features = pd.DataFrame(features, columns = col\_names)

Look at the head of the standardized dataframe:



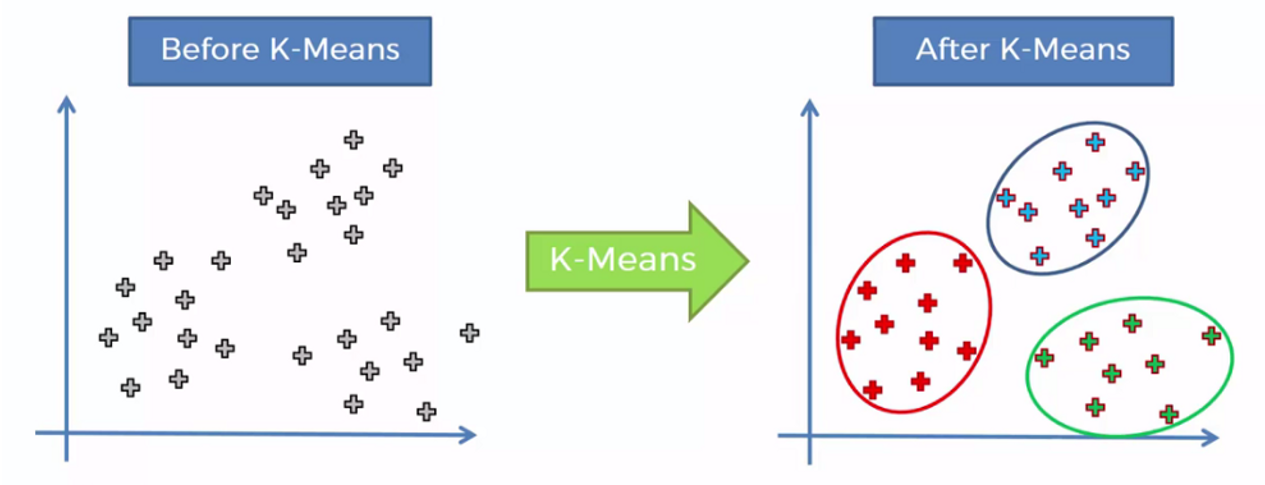
Great! We have now completed the data preparation stage and can finally start building the segmentation model.

Step 4: Building The Customer Segmentation Model

As mentioned above, we are going to create a K-Means clustering algorithm to perform customer segmentation.

The goal of a K-Means clustering model is to segment all the data available into non-overlapping sub-groups that are distinct from each other.

Here is a simple visual representation of how K-Means clustering groups a dataset into different segments:



When building a clustering model, we need to decide how many segments we want to group the data into. This is achieved by [a heuristic called the elbow method](https://iopscience.iop.org/article/10.1088/1757-899X/336/1/012017).

We will create a loop and run the K-Means algorithm from 1 to 10 clusters. Then, we can plot model results for this range of values and select the elbow of the curve as the number of clusters to use.

Run the following lines of code to achieve this:

**import** matplotlib.pyplot **as** plt

**from** sklearn.datasets **import** make\_blobs

**from** sklearn.cluster **import** KMeans

**from** sklearn.metrics **import** silhouette\_score

**from** sklearn.decomposition **import** PCA

**from** mpl\_toolkits.mplot3d **import** Axes3D

SSE = []

**for** cluster **in** range(1,10):

kmeans = KMeans(n\_clusters = cluster, init='k-means++')

kmeans.fit(scaled\_features)

SSE.append(kmeans.inertia\_)

*# converting the results into a dataframe and plotting them*

frame = pd.DataFrame({'Cluster':range(1,10), 'SSE':SSE})

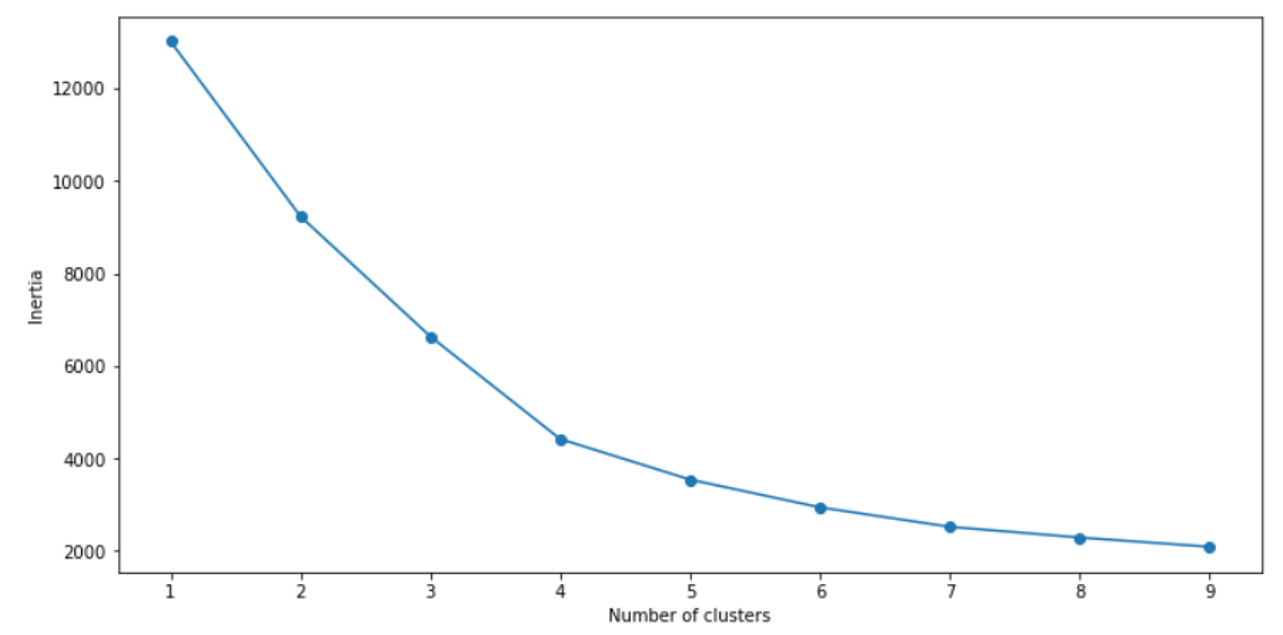
plt.figure(figsize=(12,6))

plt.plot(frame['Cluster'], frame['SSE'], marker='o')

plt.xlabel('Number of clusters')

plt.ylabel('Inertia')

Here are the results of the lines of code above:



The “elbow” of this graph is the point of inflection on the curve, and in this case is at the 4-cluster mark.

This means that the optimal number of clusters to use in this K-Means algorithm is 4. Let’s now build the model with 4 clusters:

*# First, build a model with 4 clusters*

kmeans = KMeans( n\_clusters = 4, init='k-means++')

kmeans.fit(scaled\_features)

To evaluate the performance of this model, we will use a metric called the silhouette score. This is a coefficient value that ranges from -1 to +1. A higher silhouette score is indicative of a better model.

print(silhouette\_score(scaled\_features, kmeans.labels\_, metric='euclidean'))

The silhouette coefficient of this model is **0.44,**indicating reasonable cluster separation.

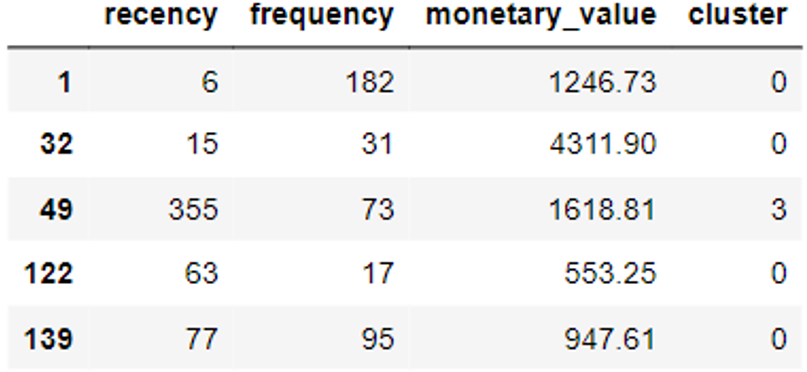
**Step 5: Segmentation Model Interpretation and Visualization**

Now that we have built our segmentation model, we need to assign clusters to each customer in the dataset:

pred = kmeans.predict(scaled\_features)

frame = pd.DataFrame(new\_df)

frame['cluster'] = pred

Let’s look at the head of the new dataframe we just created:

Then we must [visualize our data](https://365datascience.com/trending/chart-types-and-how-to-select-the-right-one/) to identify the distinct traits of customers in each segment:

avg\_df = frame.groupby(['cluster'], as\_index=False).mean()

**for** i **in** list1:

sns.barplot(x='cluster',y=str(i),data=avg\_df)

plt.show()

The codes above will render the following charts:

