CUSTOMER SEGMENTATION USING DATA SCIENCE

Step 1: Prerequisites for Building a Customer Segmentation Model

In this tutorial, we will be using an E-Commerce Dataset 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 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

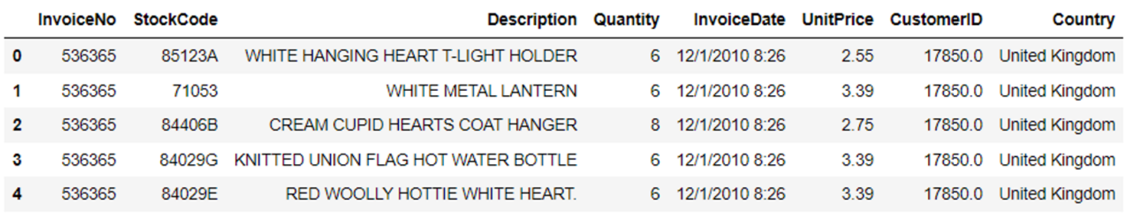
Before starting any data science project, it is vital to explore the dataset and understand each variable.

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()
```



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

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 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 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(me
thod='min').astype(int)
df_rec = df[df['rank']==1]
```

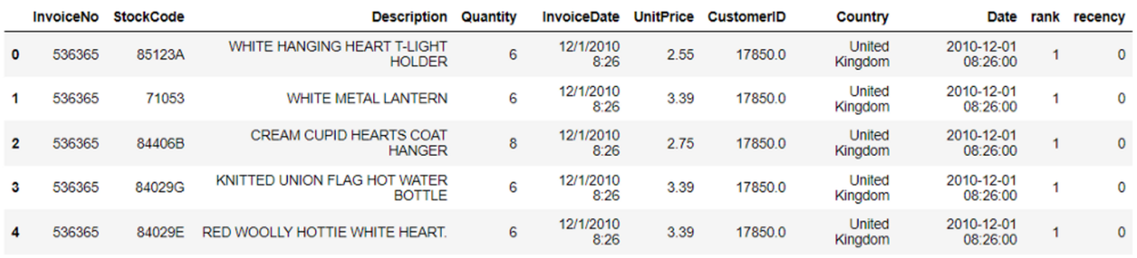
In the dataframe we just created, we only kept rows with the most recent date for each customer. We now need to rank every customer based on what time they last bought something and assign a recency score to them.

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:



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

Frequency

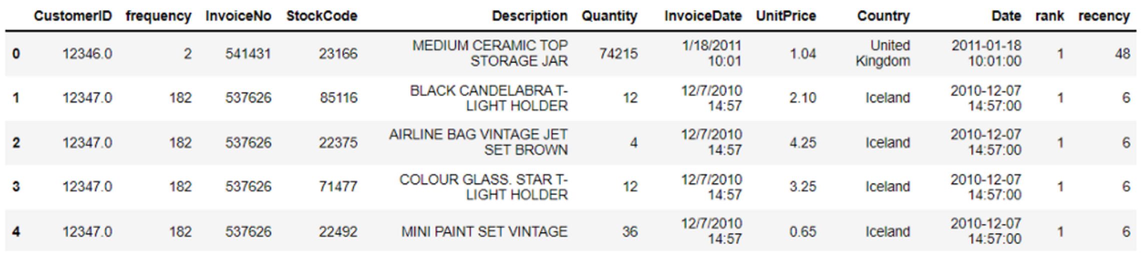
Now, let's calculate frequency—how many times has each customer made a purchase on the platform:

```
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:



	CustomerID	frequency	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	Country	Date	rank	recency
0	12346.0	2	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	1/18/2011 10:01	1.04	United Kingdom	2011-01-18 10:01:00	1	48
1	12347.0	182	537626	85116	BLACK CANDELABRA T-LIGHT HOLDER	12	12/7/2010 14:57	2.10	Iceland	2010-12-07 14:57:00	1	6
2	12347.0	182	537626	22375	AIRLINE BAG VINTAGE JET SET BROWN	4	12/7/2010 14:57	4.25	Iceland	2010-12-07 14:57:00	1	6
3	12347.0	182	537626	71477	COLOUR GLASS. STAR T-LIGHT HOLDER	12	12/7/2010 14:57	3.25	Iceland	2010-12-07 14:57:00	1	6
4	12347.0	182	537626	22492	MINI PAINT SET VINTAGE	36	12/7/2010 14:57	0.65	Iceland	2010-12-07 14:57:00	1	6

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

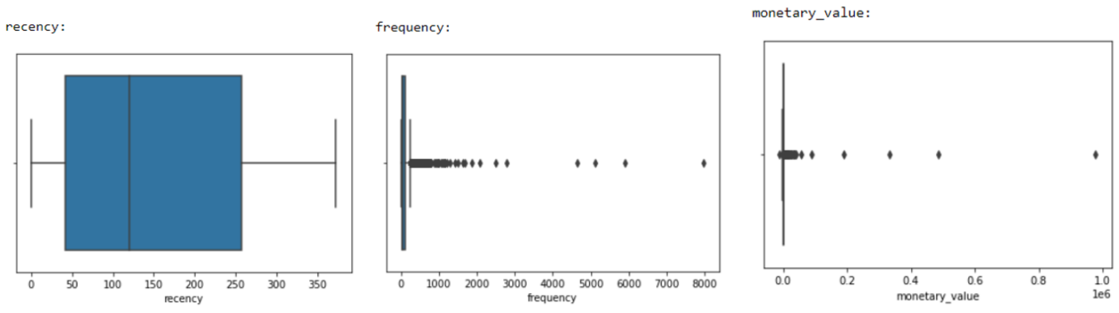
We have successfully derived three meaningful variables from the raw, uninterpretable transaction data we started out with.

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:



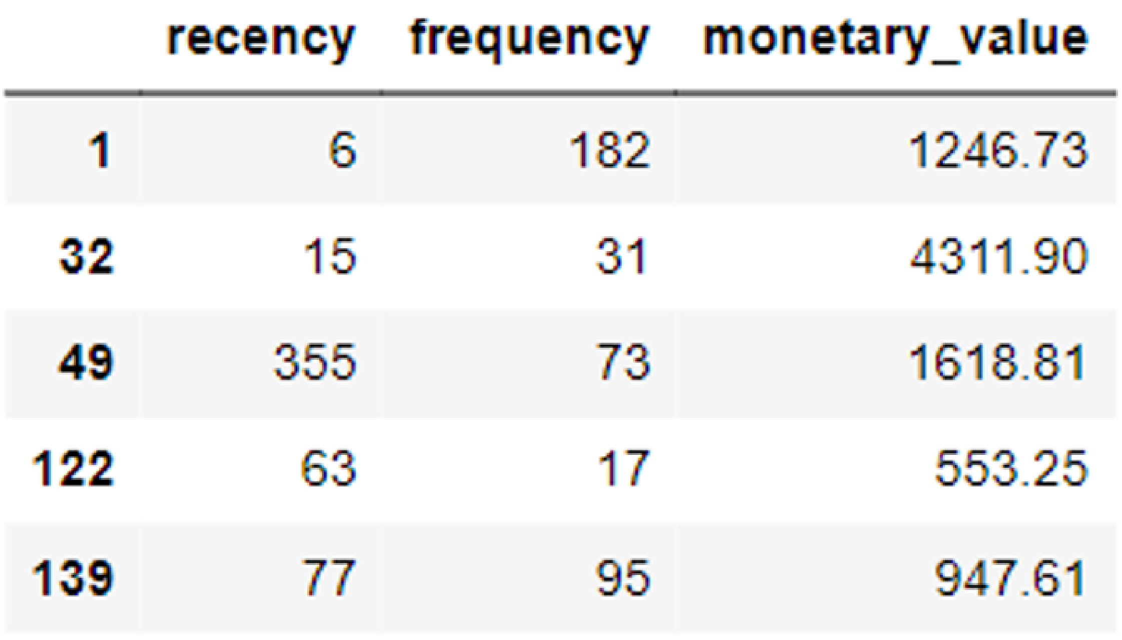
Observe that “recency” is the only variable with no visible outliers. “Frequency” and “monetary_value”, on the other hand, have many outliers that must be removed before we proceed to build the model.

To identify outliers, we will compute a measurement called a Z-Score. 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):

```python
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:



	recency	frequency	monetary_value
1	6	182	1246.73
32	15	31	4311.90
49	355	73	1618.81
122	63	17	553.25
139	77	95	947.61

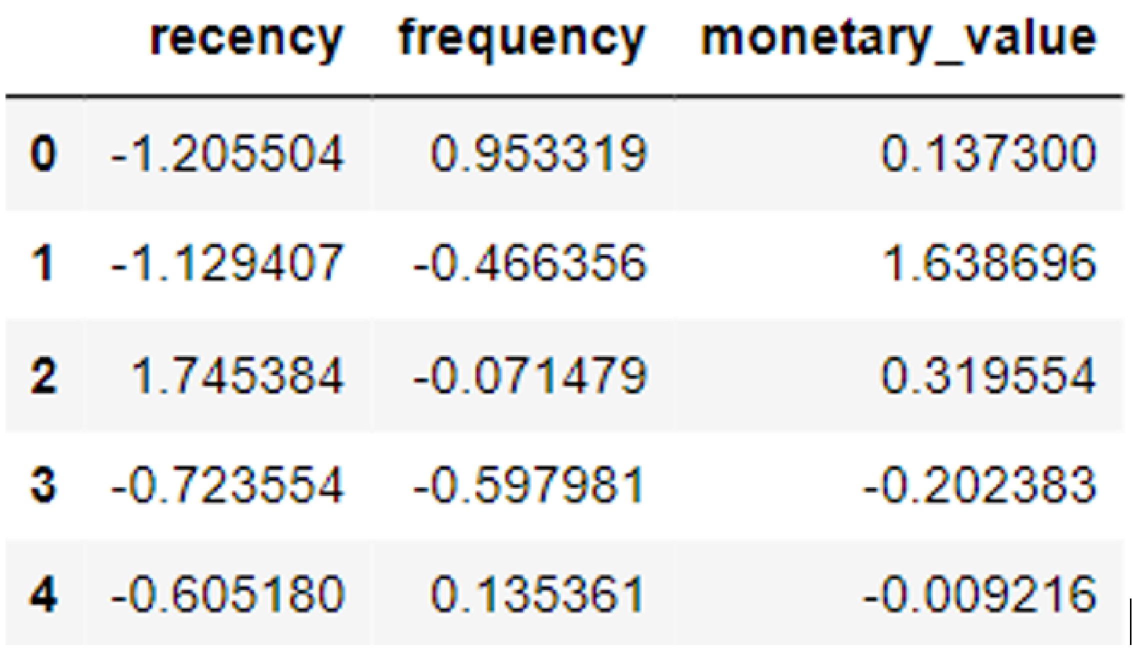
Standardization

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:



	recency	frequency	monetary_value
0	-1.205504	0.953319	0.137300
1	-1.129407	-0.466356	1.638696
2	1.745384	-0.071479	0.319554
3	-0.723554	-0.597981	-0.202383
4	-0.605180	0.135361	-0.009216

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