The aim of this project is to perform customer segmentation on a dataset of e-commerce transactions. After dividing customers into segments, linear regression is performed to predict the number of orders placed by each customer depending on a variety of different variables.

The data set contains transactional data of a online retailer that sells unique all occasion gifts to wholesales and end users. The data set variabled are explained below:

InvoiceNo: Nominal. Transaction unique identifier. If this code starts with the letter 'C', the order was cancelled

StockCode: Nominal. Product unique identifier

Description: Nominal. Product name

Quantity: Numeric. The quantities of each product per transactio.

InvoiceDate: Numeric. Invice Date and time. Numeric, the day and time when each transaction was generated.

UnitPrice: Numeric. Unit price. Numeric, Product price per unit in sterling.

CustomerID: Nominal. Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.

Country: Nominal. Country name. Nominal, the name of the country where each customer resides. The project is divided into four main sections:

Introduction Data pre-processing Data visualization Data analysis and discussion of results Conclusion and recommendations for future work

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,mean_squared_error,confusion_matrix
%matplotlib inline
Data = pd.read_csv('/content/online_retail_problem2.csv',encoding= 'unicode_escape')
Data.head()
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12-01-2010	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	12-01-2010	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12-01-2010	2.75	17850.0

```
#Formatting Date/Time
Data['InvoiceDate'] = pd.to_datetime(Data['InvoiceDate'])

#Strings
Data['Description'] = Data['Description'].str.replace('.','').str.upper().str.strip()
Data['Description'] = Data['Description'].replace('\s+',' ',regex = True)
```

```
Data['InvoiceNo'] = Data['InvoiceNo'].astype(str).str.upper()
Data['StockCode'] = Data['StockCode'].str.upper()
Data['Country'] = Data['Country'].str.upper()
Data.head()
```

```
<ipython-input-2-6c255a688f75>:5: FutureWarning: The default value of regex will c
  Data['Description'] = Data['Description'].str.replace('.','').str.upper().str.st
```

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01	2.75	17850.0
			KNITTED UNION				

```
# unnecessary transaction
#Listing Some Irrelevant StockCodes
Irrelevant = Data['StockCode'].unique()
Irrelevant.sort()
print('Irrelevant Transactions: \n',Irrelevant[::-1][:4])
#Quantity and UnitPrice Summary
Data.describe().iloc[:,:2]
```

```
Irrelevant Transactions:
 ['S' 'POST' 'PADS' 'M']
```

	Quantity	UnitPrice
count	541909.000000	541909.000000
mean	9.552250	4.611114
std	218.081158	96.759853
min	-80995.000000	-11062.060000
25%	1.000000	1.250000
50%	3.000000	2.080000
75%	10.000000	4.130000
max	80995.000000	38970.000000

```
import  scipy as sp, scipy.stats
```

```
#Outliers and Irrelevant Values
#Dropping all stockcodes that contain only strings
CodeTypes = list(map(lambda codes: any(char.isdigit() for char in codes), Data['StockCode']))
IrrelevantCodes = [i for i,v in enumerate(CodeTypes) if v == False]
Data.drop(IrrelevantCodes , inplace = True)
#Removing Outliers Based on Z-score
Data = Data[(np.abs(sp.stats.zscore(Data['UnitPrice']))<3) & (np.abs(sp.stats.zscore(Data['Quantity']))<5)]
```

```
# Missing & Incorrect Values
Data.drop(Data[(Data.Quantity>0) & (Data.InvoiceNo.str.contains('C') == True)].index, inplace = True)
Data.drop(Data[(Data.Quantity<0) & (Data.InvoiceNo.str.contains('C') == False)].index, inplace = True)
Data.drop(Data[Data.Description.str.contains('?',regex=False) == True].index, inplace = True)
Data.drop(Data[Data.UnitPrice == 0].index, inplace = True)

for index,value in Data.StockCode[Data.Description.isna()==True].items():
    if pd.notna(Data.Description[Data.StockCode == value]).sum() != 0:
        Data.Description[index] = Data.Description[Data.StockCode == value].mode()[0]
```

```
    else:
        Data.drop(index = index, inplace = True)

Data['Description'] = Data['Description'].astype(str)


#Incorrect Prices
StockList = Data.StockCode.unique()
CalculatedMode = map(lambda x: Data.UnitPrice[Data.StockCode == x].mode()[0],StockList)
StockModes = list(CalculatedMode)
for i,v in enumerate(StockList):
    Data.loc[Data['StockCode']== v, 'UnitPrice'] = StockModes[i]
```

There are also some incorrect customer IDs that for two different countries we have the same customer ID. We will fix the duplicate values by grouping the dataframe by 'CustomerID' and if any customer belongs to more than two countries, we replace the incorrect value with the mode value of the customer's country.

```
#Customers with Different Countries
Customers = Data.groupby('CustomerID')['Country'].unique()
Customers.loc[Customers.apply(lambda x:len(x)>1)]
```

```
CustomerID
12370.0           [CYPRUS, AUSTRIA]
12394.0          [BELGIUM, DENMARK]
12417.0            [BELGIUM, SPAIN]
12422.0    [AUSTRALIA, SWITZERLAND]
12429.0          [DENMARK, AUSTRIA]
12431.0        [AUSTRALIA, BELGIUM]
12455.0             [CYPRUS, SPAIN]
12457.0       [SWITZERLAND, CYPRUS]
Name: Country, dtype: object
```

```
#Fixing Duplicate CustomerIDs
for i,v in Data.groupby('CustomerID')['Country'].unique().items():
    if len(v)>1:
        Data.Country[Data['CustomerID'] == i] = Data.Country[Data['CustomerID'] == i].mode()[0]

#Adding Desired Features
Data['FinalPrice'] = Data['Quantity']*Data['UnitPrice']
Data['InvoiceMonth'] = Data['InvoiceDate'].apply(lambda x: x.strftime('%B'))
Data['Day of week'] = Data['InvoiceDate'].dt.day_name()
```

```
<ipython-input-9-71485b1c6be3>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
  Data.Country[Data['CustomerID'] == i] = Data.Country[Data['CustomerID'] == i].mode()[0]
```

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:

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

For example customer segmentation, in particular, means grouping customers together based on similar features or properties.

Now there's one thing to note is when grouping customers based on properties: the properties you choose to group the customers must be relevant to the criteria based on which you want to group them.

To segmenting customer, there are some metrics that we can use, such as when the customer buy the product for last time, how frequent the customer buy the product, and how much the customer pays for the product. We will call this segmentation as RFM segmentation.

To make the RFM table, we can create these columns, such as Recency, Frequency, and MonetaryValue column. To get the number of days for recency column, we can subtract the snapshot date with the date where the transaction occurred.

To create the frequency column, we can count how much transactions by each customer.

Lastly, to create the monetary value column, we can sum all transactions for each customer.

```
Customers.head()
```

```
CustomerID
12347.0    [ICELAND]
12348.0    [FINLAND]
12349.0      [ITALY]
12350.0     [NORWAY]
12352.0     [NORWAY]
Name: Country, dtype: object
```

```
# Sample the dataset
df_fix = Data.sample(10000, random_state = 42)
```

```
# Convert to show date only
from datetime import datetime
df_fix["InvoiceDate"] = df_fix["InvoiceDate"].dt.date
# Create TotalSum colummn
df_fix["TotalSum"] = df_fix["Quantity"] * df_fix["UnitPrice"]
# Create date variable that records recency
import datetime
snapshot_date = max(df_fix.InvoiceDate) + datetime.timedelta(days=1)
# Aggregate data by each customer
customers = df_fix.groupby(['CustomerID']).agg({
    'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
    'InvoiceNo': 'count',
    'TotalSum': 'sum'})
# Rename columns
customers.rename(columns = {'InvoiceDate': 'Recency',
                            'InvoiceNo': 'Frequency',
                            'TotalSum': 'MonetaryValue'}, inplace=True)
```

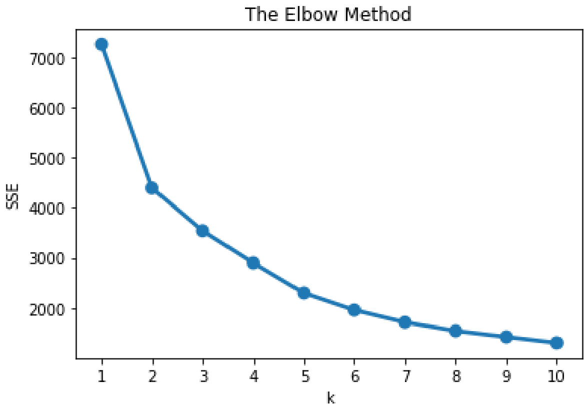
```
from scipy import stats
customers_fix = pd.DataFrame()
customers_fix["Recency"] = stats.boxcox(customers['Recency'])[0]
customers_fix["Frequency"] = stats.boxcox(customers['Frequency'])[0]
customers_fix["MonetaryValue"] = pd.Series(np.cbrt(customers['MonetaryValue'])).values
customers_fix.tail()
```

	Recency	Frequency	MonetaryValue
2414	1.276840	0.818847	3.272623
2415	6.544052	0.000000	2.924018
2416	2.780690	0.000000	2.550954
2417	4.236058	1.372703	2.899643
2418	7.382698	1.057663	5.913767

```
# Import library
from sklearn.preprocessing import StandardScaler
# Initialize the Object
scaler = StandardScaler()
# Fit and Transform The Data
scaler.fit(customers_fix)
customers_normalized = scaler.transform(customers_fix)
# Assert that it has mean 0 and variance 1
print(customers_normalized.mean(axis = 0).round(2)) # [0. -0. 0.]
print(customers_normalized.std(axis = 0).round(2)) # [1. 1. 1.]
```

```
[0. 0. 0.]
[1. 1. 1.]
```

```
from sklearn.cluster import KMeans
sse = {}
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(customers_normalized)
    sse[k] = kmeans.inertia_ # SSE to closest cluster centroid
plt.title('The Elbow Method')
plt.xlabel('k')
plt.ylabel('SSE')
sns.pointplot(x=list(sse.keys()), y=list(sse.values()))
plt.show()
```




```
model = KMeans(n_clusters=3, random_state=42)
model.fit(customers_normalized)
model.labels_.shape
```

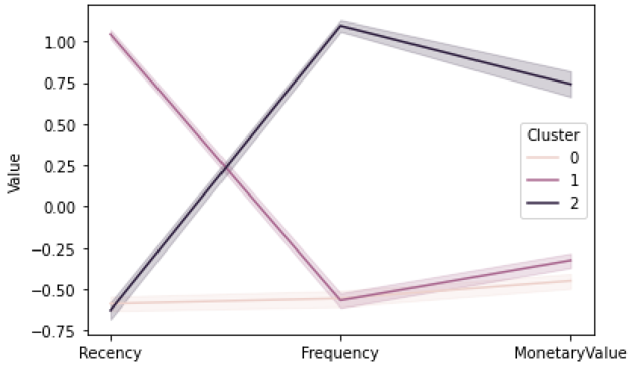
```
(2419,)
```

```
customers["Cluster"] = model.labels_
customers.groupby('Cluster').agg({
    'Recency':'mean',
    'Frequency':'mean',
    'MonetaryValue':['mean', 'count']}).round(2)
```

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	46.07	1.47	20.50	705
1	230.72	1.51	28.35	892
2	51.78	6.38	165.55	822

```
# Create the dataframe
df_normalized = pd.DataFrame(customers_normalized, columns=['Recency', 'Frequency', 'MonetaryValue'])
df_normalized['ID'] = customers.index
df_normalized['Cluster'] = model.labels_
# Melt The Data
df_nor_melt = pd.melt(df_normalized.reset_index(),
                      id_vars=['ID', 'Cluster'],
                      value_vars=['Recency','Frequency','MonetaryValue'],
                      var_name='Attribute',
                      value_name='Value')
df_nor_melt.head()
# Visualize it
sns.lineplot('Attribute', 'Value', hue='Cluster', data=df_nor_melt)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: P
  warnings.warn(
<AxesSubplot:xlabel='Attribute', ylabel='Value'>
```



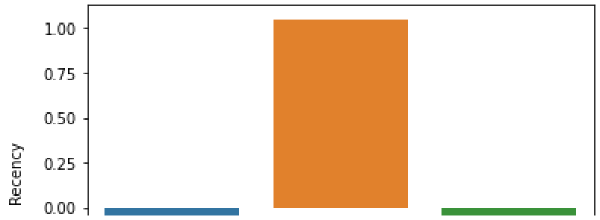
By using this plot, we know how each segment differs. It describes more than we use the summarized table.

We infer that cluster 0 is frequent, spend more, and they buy the product recently. Therefore, it could be the cluster of a loyal customer.

Then, the cluster 1 is less frequent, less to spend, but they buy the product recently. Therefore, it could be the cluster of new customer.

Finally, the cluster 2 is less frequent, less to spend, and they buy the product at the old time. Therefore, it could be the cluster of churned customers.

```
import seaborn as sns
import matplotlib.pyplot as plt
list1 = ['Recency','Frequency','MonetaryValue']


avg_df = df_normalized.groupby(['Cluster'], as_index=False).mean()
for i in list1:
    sns.barplot(x='Cluster',y=str(i),data=avg_df)
    plt.show()
```




```
Data.head()
```

oiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Coun
536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01	2.95	17850.0	UNIT KINGD
536365	71053	WHITE METAL LANTERN	6	2010-12-01	3.75	17850.0	UNIT KINGD
536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01	4.15	17850.0	UNIT KINGD
536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01	3.75	17850.0	UNIT KINGD
536365	84029E	RED WOOLLY HOTTIE WHITE HEART	6	2010-12-01	4.25	17850.0	UNIT KINGD

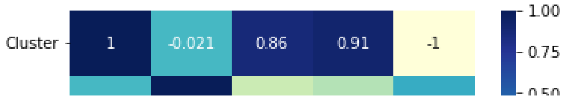
```
avg_df.head()
```

	Cluster	Recency	Frequency	MonetaryValue	ID
0	0	-0.588499	-0.557168	-0.450075	15375.273759
1	1	1.044280	-0.568493	-0.326096	15286.932735
2	2	-0.628475	1.094768	0.739879	15184.110706

PRINT CO-RELATION MATRIX

```
print(avg_df.corr())
sns.heatmap(avg_df.corr(),cmap="YlGnBu",annot=True)
```

```
               Cluster   Recency  Frequency  MonetaryValue        ID
Cluster       1.000000 -0.020942   0.863052       0.909508 -0.999045
Recency      -0.020942  1.000000  -0.523079      -0.434642  0.064606
Frequency     0.863052 -0.523079   1.000000       0.994922 -0.884298
MonetaryValue 0.909508 -0.434642   0.994922       1.000000 -0.926802
ID           -0.999045  0.064606  -0.884298      -0.926802  1.000000
<AxesSubplot:>
```



To gain even further insight into customer behavior, we can dig deeper in the relationship between RFM variables.

RFM model can be used in conjunction with certain predictive models like K-means clustering, Logistic Regression and Recommendation Engines to produce better informative results on customer behavior.

We will go for K-means since it has been widely used for Market Segmentation and it offers the advantage of being simple to implement.

Double-click (or enter) to edit

PCA Applying PCA to reduce the the dimensions and the correlation between Frequency and Monetary features.

```
features = avg_df.columns
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
avg_df = pd.DataFrame(pt.fit_transform(avg_df))
avg_df.columns = features
avg_df.head()
```

	Cluster	Recency	Frequency	MonetaryValue	ID
0	-1.267550	-0.640237	-0.687479	-0.958626	1.211594
1	0.090648	1.412168	-0.726555	-0.421118	0.025892
2	1.176903	-0.771931	1.414034	1.379745	-1.237485

```
sc = StandardScaler()
rfm_scaled = sc.fit_transform(avg_df)
rfm_scaled
```

```
array([[-1.26755013, -0.64023721, -0.68747905, -0.95862629,  1.21159382],
       [ 0.09064754,  1.41216816, -0.72655455, -0.42111827,  0.02589155],
       [ 1.17690258, -0.77193096,  1.4140336 ,  1.37974456, -1.23748537]])
```

```
from sklearn.decomposition import PCA
pca = PCA()
pca_tranformed_data = pca.fit_transform(rfm_scaled)
```

```
pca.components_
```

```
array([[ 0.47833602, -0.16894809,  0.49251335,  0.50888869, -0.49103371],
       [ 0.32005763,  0.88358345, -0.23625905, -0.01666306, -0.24646988],
       [-0.70245395,  0.02774446, -0.17789494,  0.20840285, -0.65628526]])
```

```
pca.explained_variance_
```

```
array([5.79039529e+00, 1.70960471e+00, 1.13308878e-32])
```

MODEL TRAINING

```
X = avg_df.copy()
pca = PCA(n_components = 2)
df_pca = pca.fit_transform(X)
```

```
df_pca = pd.DataFrame(df_pca)
df_pca.head(5)
```

	0	1
0	-1.919508	-1.091617
1	-0.780077	1.449072
2	2.699585	-0.357455

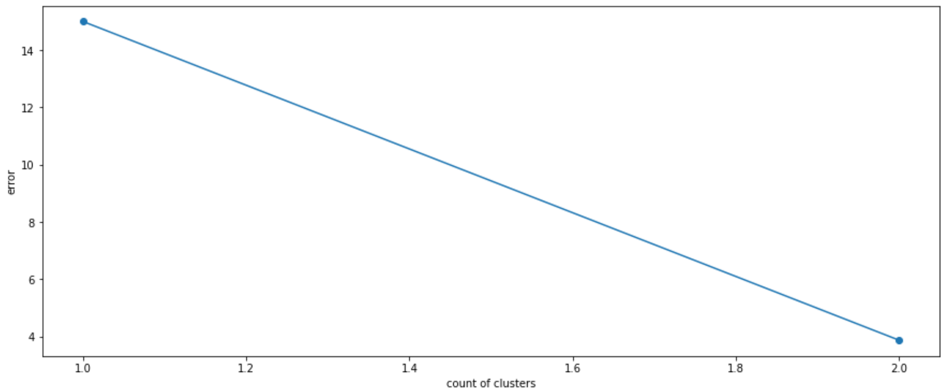
```
X = df_pca.copy()
```

```
from sklearn.cluster import KMeans

cluster_range = range(1, 3)
cluster_errors = []
cluster_sil_scores = []


for num in cluster_range:
    clusters = KMeans(num, n_init = 100,init='k-means++',random_state=0)
    clusters.fit(X)
    # capture the cluster lables
    labels = clusters.labels_
    # capture the centroids
    centroids = clusters.cluster_centers_
    # capture the intertia
    cluster_errors.append( clusters.inertia_ )
clusters_df = pd.DataFrame({ "num_clusters":cluster_range, "cluster_errors": cluster_errors} )
clusters_df[0:10]
```

	num_clusters	cluster_errors
0	1	15.0000
1	2	3.8767

```
plt.figure(figsize=(15,6))
plt.plot(clusters_df["num_clusters"],clusters_df["cluster_errors"],marker = 'o')
plt.xlabel('count of clusters')
plt.ylabel('error')
```

```
Text(0, 0.5, 'error')
```



Inferences: We observe from the elbow plot a sharp bend after the number of clusters increase by 2. Silhoutte Score is also the highest for 2 clusters.

But, there is also a significant reduce in cluster error as number of clusters increase from 2 to 4 and after 4, the reduction is not much.

So, we will choose n_clusters = 4 to properly segment our customers.

Summary

The work described in this notebook is based on a database providing details on purchases made on an E-commerce platform over a period of one year. Each entry in the dataset describes the purchase of a product, by a particular customer and at a given date.

Summary The work described in this notebook is based on a database providing details on purchases made on an E-commerce platform over a period of one year. Each entry in the dataset describes the purchase of a product, by a particular customer and at a given date. Given the available information, I decided to develop a classifier that allows to anticipate the type of purchase that a customer will make, as well as the number of visits that he will make during a year, and this from its first visit to the E-commerce site.

The next part of the analysis consisted of some basic data visualization. This was done in order to get insights regarding the country which was using the E-commerce website the most. I used basic plots in order to show the results of my analysis. I also tried to analyse other important factors such as the Gross Purcahse by a country as well as which following description was used the most.

The final part of the analysis was the customer segmentation part. The main way to go around with this procces is to use the RFM (Recency, Frequency, Monetory) table to sort the customer in the groups. After creating the RFM table I used K-Means clustering (Elbow curve and Silhoutte scores) in order to create 4 clusters in which the customers should be Segmented. After each of the customers were segmented into their respective groups. I used models such as Logisitc Regression, KNeighborsClassifier ,DecisionTree in order the cross the accuracy of the clustering which resulted in an accuracy score 0.98. Hence, I conclude the customer segmentation was done which effective methods and high accuracy.