

**ITE 2006 - DATA MINING TECHNIQUES**

**PROJECT REVIEW**

**TITLE – CONSUMER PROFILING AND SEGMENTATION**

**Submitted by : Submitted to:**

Ritveak Dugar(16BIT0407) Lakshmi Priya G G

Joji Johnson (16BIT0195)

**PROBLEM STATEMENT: -**

Segmenting potential customers to target them in order to offer them relevant products.

**PROBLEM DESCRIPTION: -**

Customers often get confused seeing the variety of products they are surrounded by.

Even the marketers waste a lot of time and money on convincing and approaching customers who are less interested or are not in need of that product.

So in order to solve both the issues, we will try to come up with a solution using data mining and hence will get a better idea about interests and wants of individual customers.

Having a better understanding of customers will help the marketers to communicate with the targeted people more effectively, giving them the best insights about their favorable products.

**DATASET DESCRIPTION:**

|  |  |
| --- | --- |
| **Abstract**: This 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. |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Set Characteristics:** | Multivariate, Sequential, Time-Series | **Number of Instances:** | 541909 | **Area:** | Business |
| **Attribute Characteristics:** | Integer, Real | **Number of Attributes:** | 8 | **Date Donated** | 2015-11-06 |
| **Associated Tasks:** | Classification, Clustering | **Missing Values?** | N/A | **Number of Web Hits:** | 253549 |

**Source:**

Dr Daqing Chen, Director: Public Analytics group. chend**'@'** lsbu.ac.uk, School of Engineering, London South Bank University, London SE1 0AA, UK.

**Attribute Information:**

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: Invice 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.

**ALGORITHM:**

For Customer Profiling and Segmentation, we will be using Association Data Mining rule(Apriori) and also K-means Clustering

**Apriori Algorithm:**

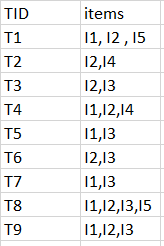
**Apriori algorithm** is used for finding frequent itemsets in a dataset for Boolean association rule. Name of algorithm is Apriori is because it uses prior knowledge of frequent item set properties. We apply a iterative approach or level-wise search where k-frequent itemsets are used to find k+1 itemsets.

To improve the efficiency of level-wise generation of frequent itemsets an important property is used called *Apriori property* which helps by reducing the search space.

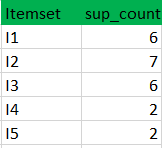
**Apriori Property –**  
All nonempty subset of frequent itemset must be frequent. The key concept of Apriori algorithm is its anti-monotonicity of support measure. Apriori assumes that

All subsets of a frequent itemset must be frequent(Aprioripropertry).  
If a itemset is infrequent all its supersets will be infrequent.

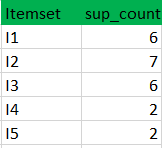
Before we start understanding algorithm go through some definitions which are explained in my previous post.  
Consider the following dataset and we will find frequent itemsets and generate association rules on this.

  
minimum support count is 2  
minimum confidence is 60%

**Step-1:** K=1  
(I) Create a table containing support count of each item present in dataset – Called **C1(candidate set)**

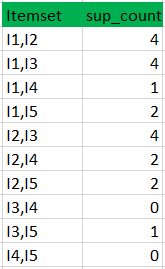


(II) compare candidate set item’s support count with minimum support count(here min\_support=2 if support\_count of candidate set items is less than min\_support then remove those items) this gives us itemset L1.

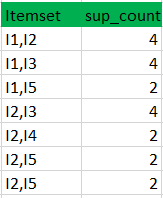


**Step-2:** K=2

* Generate candidate set C2 using L1 (this is called join step). Condition of joining is Lk-1 and Lk-1 is that it should have (K-2) elements in common.
* Check all subsets of a itemset are frequent or not and if not frequent remove that itemset.(Example subset of{I1, I2} are {I1}, {I2} they are frequent.Check for each itemset)
* Now find support count of these itemsets by searching in dataset.

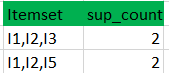


(II) compare candidate (C2) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L2.

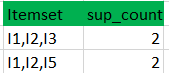


**Step-3:**

* + Generate candidate set C3 using L2 (join step). Condition of joining Lk-1 and Lk-1 is it should have (K-2) elements in common. So here for L2 first element should match.  
    So itemset generated by joining L2 is {I1, I2, I3}{I1, I2, I5}{I1, I3, i5}{I2, I3, I4}{I2, I4, I5}{I2, I3, I5}
  + Check all subsets of these itemsets are frequent or not and if not remove that itemset.(Here subset of {I1, I2, I3} are {I1, I2}{I2, I3}{I1, I3} which are frequent. For {I2, I3, I4} subset {I3, I4} is not frequent so remove this. Similarly check for every itemset)
  + find support count of these remaining itemset by searching in dataset.



(II) Compare candidate (C3) support count with minimum support count(here min\_support=2 if support\_count of candidate set item is less than min\_support then remove those items) this gives us itemset L3.



**Step-4:**

* + Generate candidate set C4 using L3 (join step). Condition of joining Lk-1 and Lk-1 (K=4)is these should have (K-2) elements in common. So here for L3 first 2 element(items) should match.
  + Check all subsets of these itemsets are frequent or not(Here itemset formed by joining L3 is {I1, I2, I3, I5} so its subset contain {I1, I3, I5} which is not frequent). so no itemset in C4
  + We stop here because no frequent itemset are found frequent further

Thus we discovered all frequent item-sets now generation of strong association rule comes into picture. For that we need to calculate confidence of each rule.

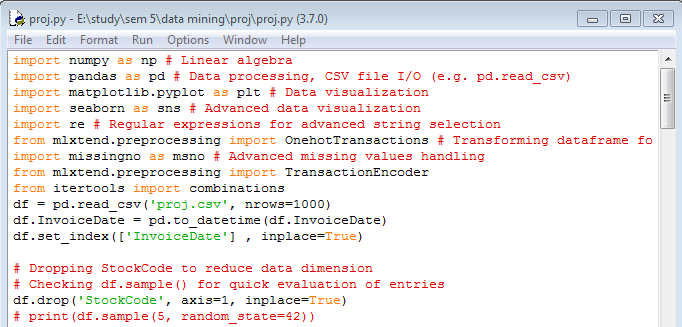
**Confidence –**  
Confidence(A->B)=Support\_count(A∪B)/Support\_count(A)

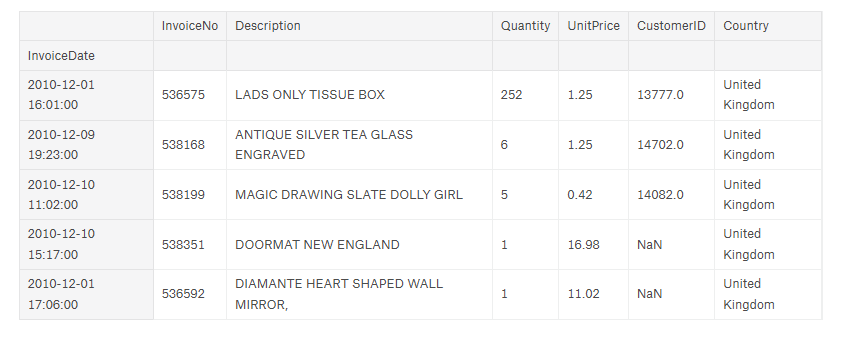
So here By taking example of any frequent itemset we will show rule generation.  
Itemset {I1, I2, I3} //from L3  
SO rules can be  
[I1^I2]=>[I3] //confidence = sup(I1^I2^I3)/sup(I1^I2) = 2/4\*100=50%  
[I1^I3]=>[I2] //confidence = sup(I1^I2^I3)/sup(I1^I3) = 2/4\*100=50%  
[I2^I3]=>[I1] //confidence = sup(I1^I2^I3)/sup(I2^I3) = 2/4\*100=50%  
[I1]=>[I2^I3] //confidence = sup(I1^I2^I3)/sup(I1) = 2/6\*100=33%  
[I2]=>[I1^I3] //confidence = sup(I1^I2^I3)/sup(I2) = 2/7\*100=28%  
[I3]=>[I1^I2] //confidence = sup(I1^I2^I3)/sup(I3) = 2/6\*100=33%

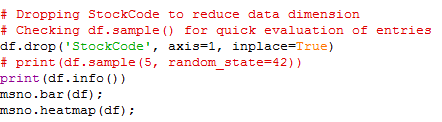
So if minimum confidence is 50 % first 3 rules can be considered strong association rules.

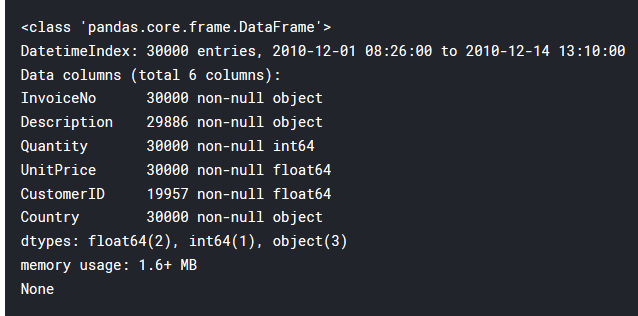
**CODE SNIPPET**:

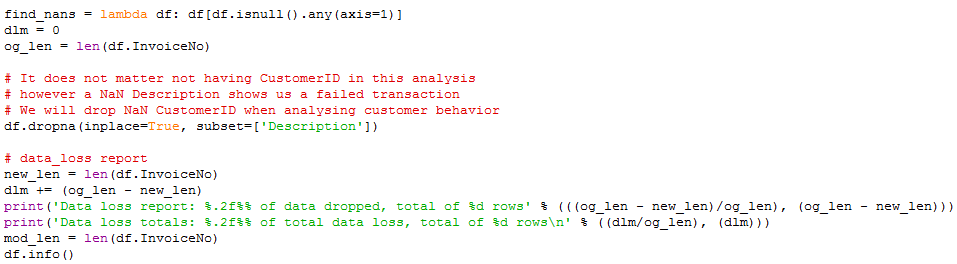
Pre-Processing:

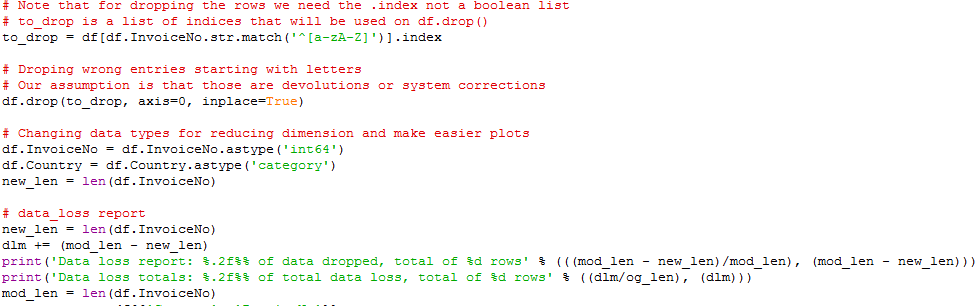
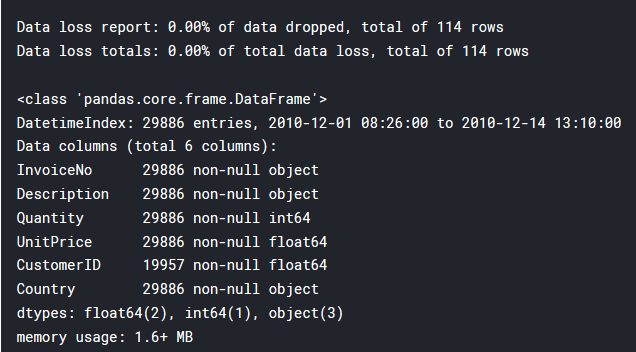


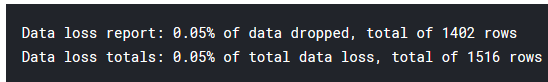




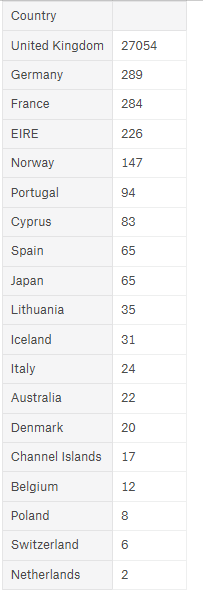




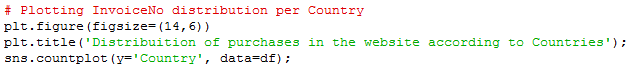


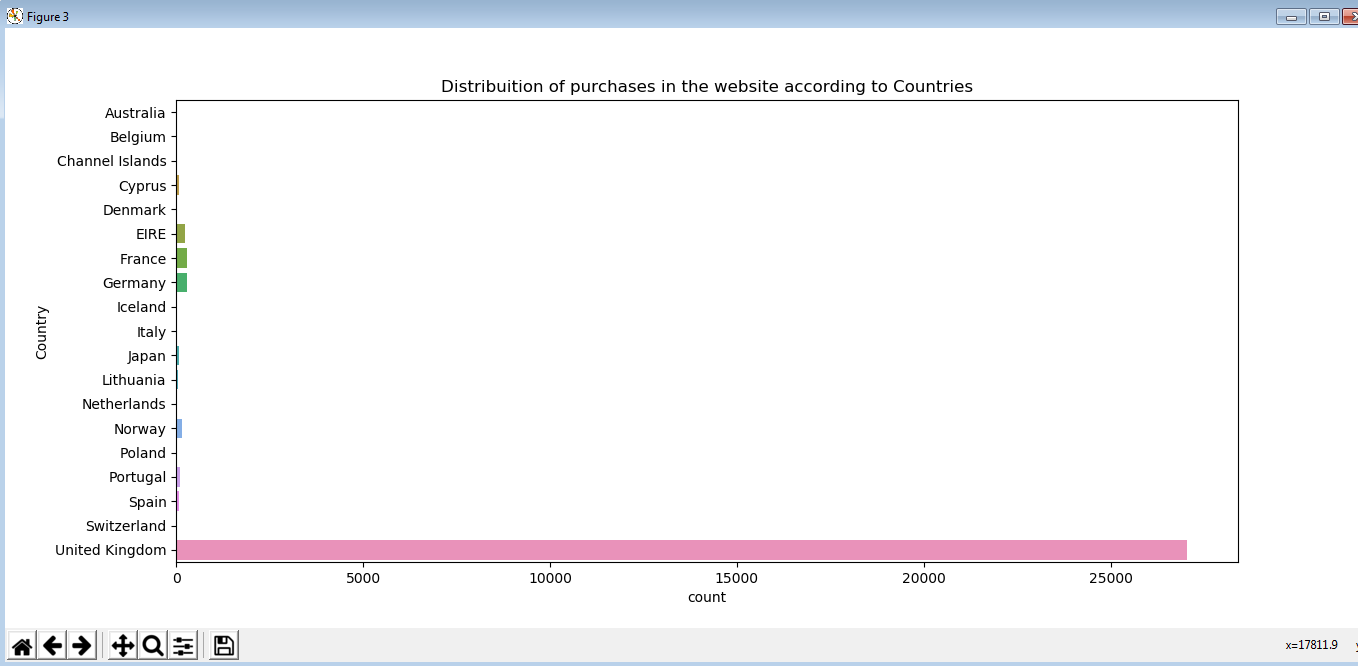


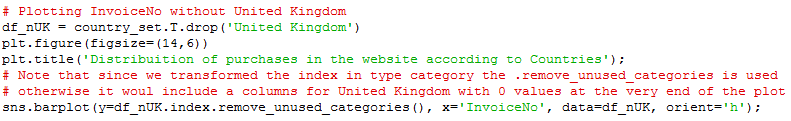


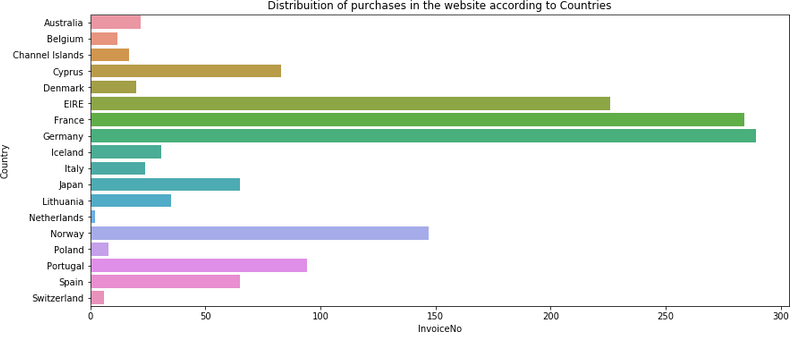


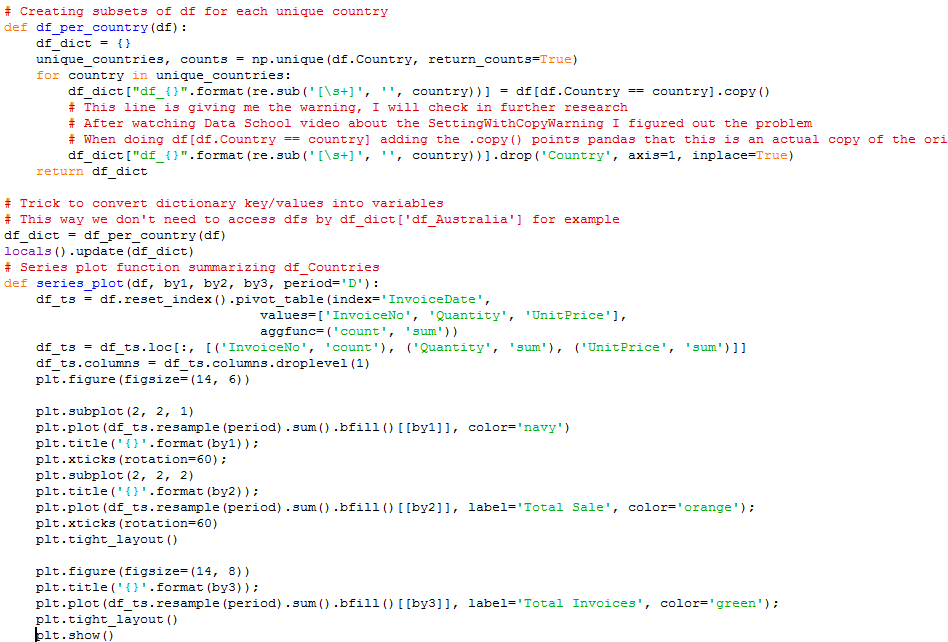
**DATA VISUALISATION**



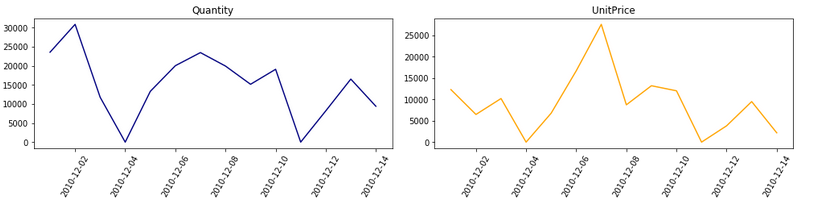


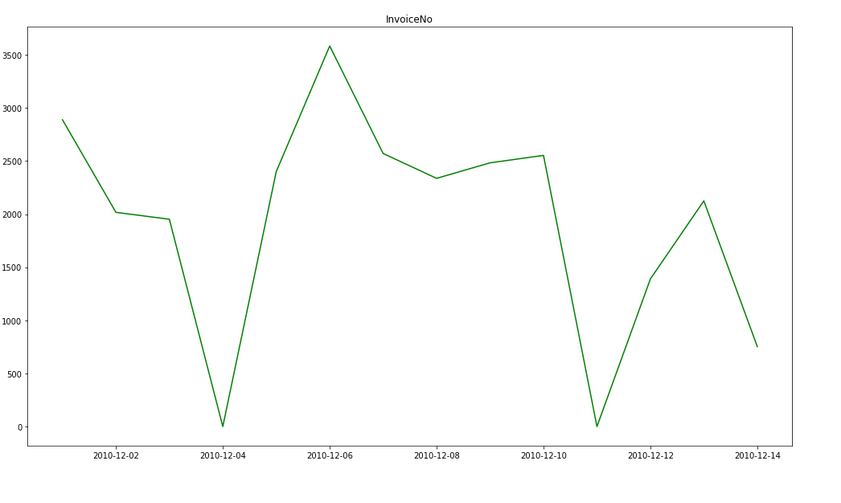












**Algorithm code:**

def apriori(df, min\_support=0.5, use\_colnames=False, max\_len=None, n\_jobs=1):

allowed\_val = {0, 1, True, False}

unique\_val = np.unique(df.values.ravel())

for val in unique\_val:

if val not in allowed\_val:

s = ('The allowed values for a DataFrame'

' are True, False, 0, 1. Found value %s' % (val))

raise ValueError(s)

is\_sparse = hasattr(df, "to\_coo")

if is\_sparse:

X = df.to\_coo().tocsc()

support = np.array(np.sum(X, axis=0) / float(X.shape[0])).reshape(-1)

else:

X = df.values

support = (np.sum(X, axis=0) / float(X.shape[0]))

ary\_col\_idx = np.arange(X.shape[1])

support\_dict = {1: support[support >= min\_support]}

itemset\_dict = {1: ary\_col\_idx[support >= min\_support].reshape(-1, 1)}

max\_itemset = 1

rows\_count = float(X.shape[0])

if max\_len is None:

max\_len = float('inf')

while max\_itemset and max\_itemset<max\_len:

next\_max\_itemset = max\_itemset + 1

combin = generate\_new\_combinations(itemset\_dict[max\_itemset])

frequent\_items = []

frequent\_items\_support = []

if is\_sparse:

all\_ones = np.ones((X.shape[0], next\_max\_itemset))

for c in combin:

if is\_sparse:

together = np.all(X[:, c] == all\_ones, axis=1)

else:

together = X[:, c].all(axis=1)

support = together.sum() / rows\_count

if support >= min\_support:

frequent\_items.append(c)

frequent\_items\_support.append(support)

if frequent\_items:

itemset\_dict[next\_max\_itemset] = np.array(frequent\_items)

support\_dict[next\_max\_itemset] = np.array(frequent\_items\_support)

max\_itemset = next\_max\_itemset

else:

max\_itemset = 0

all\_res = []

for k in sorted(itemset\_dict):

support = pd.Series(support\_dict[k])

itemsets = pd.Series([frozenset(i) for i in itemset\_dict[k]])

res = pd.concat((support, itemsets), axis=1)

all\_res.append(res)

res\_df = pd.concat(all\_res)

res\_df.columns = ['support', 'itemsets']

if use\_colnames:

mapping = {idx: item for idx, item in enumerate(df.columns)}

res\_df['itemsets'] = res\_df['itemsets'].apply(lambda x: frozenset([

mapping[i] for i in x]))

res\_df = res\_df.reset\_index(drop=True)

return res\_df

defassociation\_rules(df, metric="confidence",

min\_threshold=0.8, support\_only=False):

# check for mandatory columns

if not all(col in df.columns for col in ["support", "itemsets"]):

raise ValueError("Dataframe needs to contain the\

columns 'support' and 'itemsets'")

def conviction\_helper(sAC, sA, sC):

confidence = sAC/sA

conviction = np.empty(confidence.shape, dtype=float)

if not len(conviction.shape):

conviction = conviction[np.newaxis]

confidence = confidence[np.newaxis]

sAC = sAC[np.newaxis]

sA = sA[np.newaxis]

sC = sC[np.newaxis]

conviction[:] = np.inf

conviction[confidence < 1.] = ((1. - sC[confidence < 1.]) /

(1. - confidence[confidence < 1.]))

return conviction

# metrics for association rules

metric\_dict = {

"antecedent support": lambda \_, sA, \_\_: sA,

"consequent support": lambda \_, \_\_, sC: sC,

"support": lambda sAC, \_, \_\_: sAC,

"confidence": lambda sAC, sA, \_: sAC/sA

}

columns\_ordered = ["antecedent support", "consequent support",

"support",

"confidence"]

# check for metric compliance

if metric not in metric\_dict.keys():

raise ValueError("Metric must be 'confidence' or 'lift', got '{}'"

.format(metric))

# get dict of {frequent itemset} -> support

keys = df['itemsets'].values

values = df['support'].values

frozenset\_vect = np.vectorize(lambda x: frozenset(x))

frequent\_items\_dict = dict(zip(frozenset\_vect(keys), values))

# prepare buckets to collect frequent rules

rule\_antecedents = []

rule\_consequents = []

rule\_supports = []

# iterate over all frequent itemsets

for k in frequent\_items\_dict.keys():

sAC = frequent\_items\_dict[k]

# to find all possible combinations

for idx in range(len(k)-1, 0, -1):

# of antecedent and consequent

for c in combinations(k, r=idx):

antecedent = frozenset(c)

consequent = k.difference(antecedent)

try:

sA = frequent\_items\_dict[antecedent]

sC = frequent\_items\_dict[consequent]

except KeyError as e:

s = (str(e) + 'You are likely getting this error'

' because the DataFrame is missing '

' antecedent and/or consequent '

' information.'

' You can try using the '

' `support\_only=True` option')

raise KeyError(s)

# check for the threshold

score = metric\_dict[metric](sAC, sA, sC)

if score >= min\_threshold:

rule\_antecedents.append(antecedent)

rule\_consequents.append(consequent)

rule\_supports.append([sAC, sA, sC])

# check if frequent rule was generated

if not rule\_supports:

return pd.DataFrame(

columns=["antecedents", "consequents"] + columns\_ordered)

else:

# generate metrics

rule\_supports = np.array(rule\_supports).T.astype(float)

df\_res = pd.DataFrame(

data=list(zip(rule\_antecedents, rule\_consequents)),

columns=["antecedents", "consequents"])

sAC = rule\_supports[0]

sA = rule\_supports[1]

sC = rule\_supports[2]

for m in columns\_ordered:

df\_res[m] = metric\_dict[m](sAC, sA, sC)

return df\_res

**OUTPUT:**

Frequent Itemlist



Association rule:



**K-MEANS ALGORITHM**

K-means is a method commonly used to automatically partition a dataset into K-groups. It is unsupervised algorithm. The objective of K-means is to minimize the total sum of the squared distance of every point to its corresponding cluster centroid. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity. The results of the *K*-means clustering algorithm are:

1. The centroids of the *K* clusters, which can be used to label new data
2. Labels for the training data (each data point is assigned to a single cluster)

Rather than defining groups before looking at the data, clustering allows you to find and analyze the groups that have formed organically. The "Choosing K" section below describes how the number of groups can be determined.

Each centroid of a cluster is a collection of feature values which define the resulting groups. Examining the centroid feature weights can be used to qualitatively interpret what kind of group each cluster represents.

**Applying K-means algorithm:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

dataset = pd.read\_excel("proj.xlsx")

dataset['TotalPrice'] = dataset['Quantity'] \* dataset['UnitPrice']

X=dataset.iloc[: , [6,8]].values

class K\_Means:

def \_\_init\_\_(self, k=3, tol=0.001, max\_iter=300):

self.k = k

self.tol = tol

self.max\_iter = max\_iter

def fit(self,data):

self.centroids = {}

for i in range(self.k):

self.centroids[i] = data[i]

for i in range(self.max\_iter):

self.classifications = {}

for i in range(self.k):

self.classifications[i] = []

for featureset in X:

distances = [np.linalg.norm(featureset-self.centroids[centroid]) for centroid in self.centroids]

classification = distances.index(min(distances))

self.classifications[classification].append(featureset)

prev\_centroids = dict(self.centroids)

for classification in self.classifications:

self.centroids[classification] = np.average(self.classifications[classification],axis=0)

optimized = True

for c in self.centroids:

original\_centroid = prev\_centroids[c]

current\_centroid = self.centroids[c]

if np.sum((current\_centroid-original\_centroid)/original\_centroid\*100.0) > self.tol:

print(np.sum((current\_centroid-original\_centroid)/original\_centroid\*100.0))

optimized = False

if optimized:

break

def predict(self,data):

distances = [np.linalg.norm(data-self.centroids[centroid]) for centroid in self.centroids]

classification = distances.index(min(distances))

return classification

clf = K\_Means()

clf.fit(X)

plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red')

plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'green')

plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 250, c = 'yellow', label='Centroids')

plt.title('Clusters of customer Invoices & Expenses')

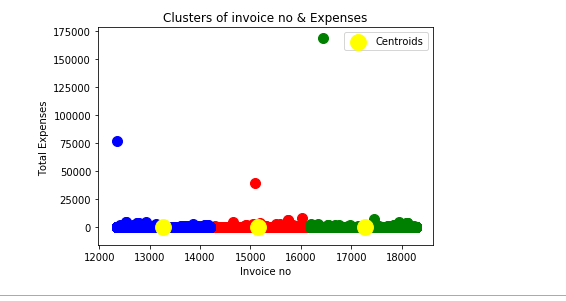
plt.title('Clusters of invoice no & Expenses')

plt.xlabel('Invoice no')

plt.ylabel('Total Expenses ')

plt.legend()

plt.show()



**FINDING BEST CUSTOMER USING CUSTOMER SEGMENTATION:**

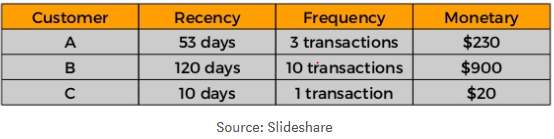
**Overview**

When it comes to finding out who your best customers are, the old RFM matrix principle is the best. RFM stands for Recency, Frequency and Monetary. It is a customer segmentation technique that uses past purchase behaviorto divide customers into groups.

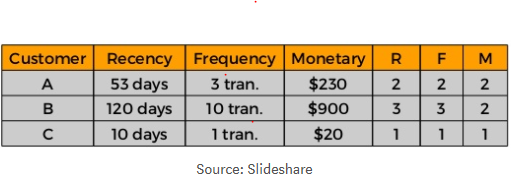
**RFM Score Calculations**

**RECENCY (R):** Days since last purchase  
**FREQUENCY (F):** Total number of purchases  
**MONETARY VALUE (M):** Total money this customer spent

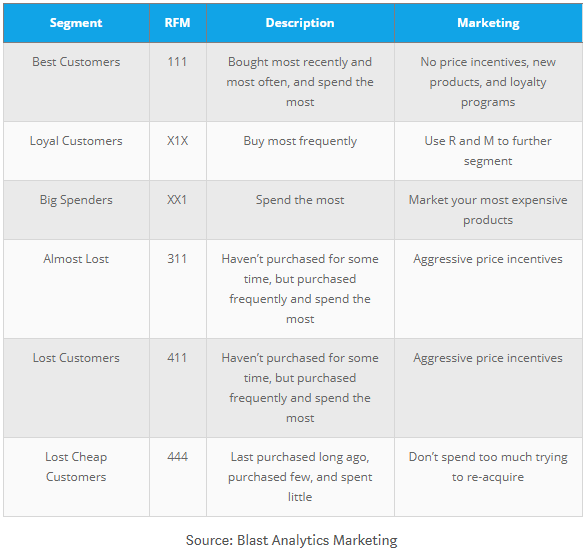
Step 1: Calculate the RFM metrics for each customer.



Step 2: Add segment numbers to RFM table.



Step 3: Sort according to the RFM scores from the best customers (score 111).



**CODE (WITH PREPROCESSING)**

import pandas as pd

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_excel("Online\_Retail.xlsx")

df.head()

df1 = df

df1.Country.nunique()

df1.Country.unique()

customer\_country=df1[['Country','CustomerID']].drop\_duplicates()

customer\_country.groupby(['Country'])['CustomerID'].aggregate('count').reset\_index().sort\_values('CustomerID', ascending=False)

df1 = df1.loc[df1['Country'] == 'United Kingdom']

df1 = df1[pd.notnull(df1['CustomerID'])]

df1 = df1[pd.notnull(df1['CustomerID'])]

df1.Quantity.min()

df1 = df1[(df1['Quantity']>0)]

df1.shape

df1.info()

def unique\_counts(df1):

for i in df1.columns:

count = df1[i].nunique()

print(i, ": ", count)

unique\_counts(df1)

df1['TotalPrice'] = df1['Quantity'] \* df1['UnitPrice']

df1['InvoiceDate'].min()

import datetime as dt

NOW = dt.datetime(2011,12,10)

df1['InvoiceDate'] = pd.to\_datetime(df1['InvoiceDate'])

rfmTable = df1.groupby('CustomerID').agg({'InvoiceDate': lambda x: (NOW - x.max()).days, 'InvoiceNo': lambda x: len(x), 'TotalPrice': lambda x: x.sum()})

rfmTable['InvoiceDate'] = rfmTable['InvoiceDate'].astype(int)

rfmTable.rename(columns={'InvoiceDate': 'recency',

'InvoiceNo': 'frequency',

'TotalPrice': 'monetary\_value'}, inplace=True)

quantiles = rfmTable.quantile(q=[0.25,0.5,0.75])

quantiles = quantiles.to\_dict()

segmented\_rfm = rfmTable

def RScore(x,p,d):

if x <= d[p][0.25]:

return 1

elif x <= d[p][0.50]:

return 2

elif x <= d[p][0.75]:

return 3

else:

return 4

def FMScore(x,p,d):

if x <= d[p][0.25]:

return 4

elif x <= d[p][0.50]:

return 3

elif x <= d[p][0.75]:

return 2

else:

return 1

segmented\_rfm['r\_quartile'] = segmented\_rfm['recency'].apply(RScore, args=('recency',quantiles,))

segmented\_rfm['f\_quartile'] = segmented\_rfm['frequency'].apply(FMScore, args=('frequency',quantiles,))

segmented\_rfm['m\_quartile'] = segmented\_rfm['monetary\_value'].apply(FMScore, args=('monetary\_value',quantiles,))

segmented\_rfm.head()

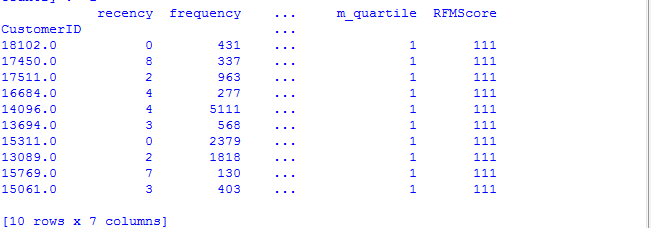
segmented\_rfm['RFMScore'] = segmented\_rfm.r\_quartile.map(str) + segmented\_rfm.f\_quartile.map(str) + segmented\_rfm.m\_quartile.map(str)

segmented\_rfm.head()

print(segmented\_rfm[segmented\_rfm['RFMScore']=='111'].sort\_values('monetary\_value', ascending=False).head(10))

**OUTPUT:**

TOP 10 CUSTOMERS:



Conclusion:

Apriori gave us more insights to the customer behavior and greatly reduces the number of database access and the searching cost as the data is very large, this greatly improved the efficiency but K-means only showed us that how frequently a customer is buying the items and generating the invoice no and the total expenses a customer can afford on a period of time. Therefore It can be concluded that the Association rule Data Mining is more preferable than other existing algorithms.