Configuration Manual

MSc Research Project
Data Analytics

Ashwini Mohan
Student ID: x19220618

School of Computing
National College of Ireland

Supervisor: Prof. Athanasios Staikopoulos

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ashwini Mohan
Student ID:	x19220618
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Prof. Athanasios Staikopoulos
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	1039
Page Count:	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Ashwini Mohan
Date:	31st January 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ashwini Mohan
x19220618

1 Introduction

This configuration handbook outlines both software and hardware requirements, as well as a step-by-step procedure for carrying out the research objective of implementing customer segmentation using RFM analysis with K-Means Clustering, multiclass classification model, and Market Basket Analysis.

2 Environment Specification and Configuration

Pre-requisite - Anaconda version 1.9.12 should already be installed with Jupyter Notebook. Installation link - `https://www.anaconda.com/products/individual#windows`

2.1 Hardware Configuration

The screenshot of hardware configuration of system details in 1 can be seen.

- Windows Edition: Windows 10 Home.
- Processor: Intel(R) Core™ i5-8250U CPU @ 1.60GHz 1.80 GHz
- Installed Memory (RAM) : 8GB
- System type: 64-bit operating System, x64-based processor

2.2 Software Requirements

The specifications for software required is detailed below:

- Programming Language - Python (version - 3.7.6)
- IDE - Jupyter Notebook - version 6.0.3
- Browser - Google Chrome

3 Environment Setup

The Jupyter Notebook is initiated from Anaconda to begin implementation execution.

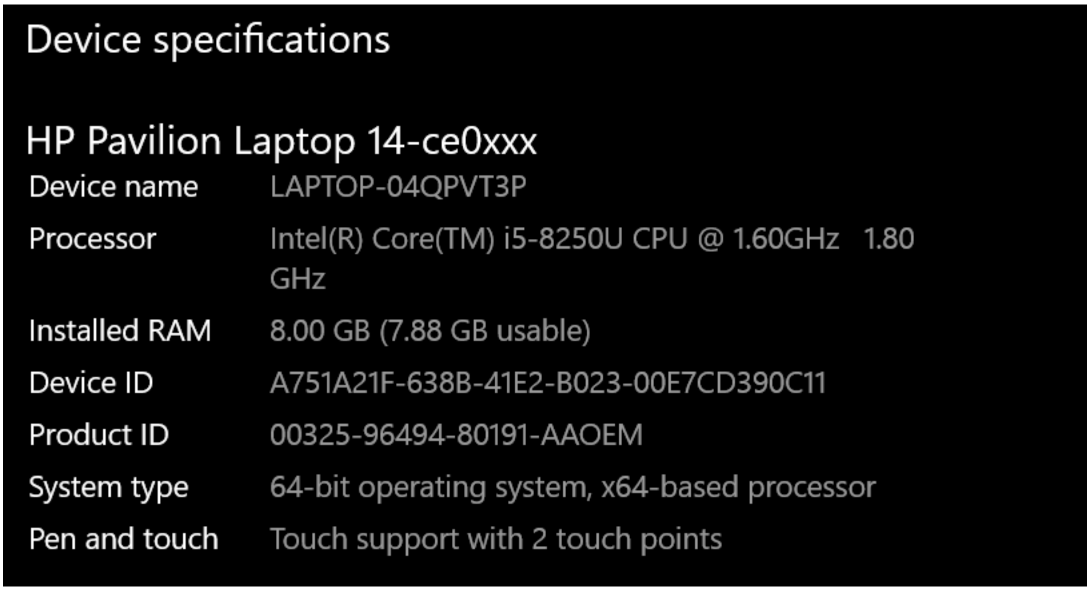


Figure 1: Windows Specification

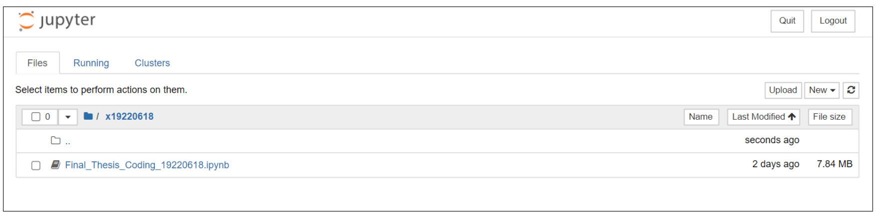


Figure 2: Jupyter Notebook on Initiation

4 Library Packages Required

Before importing any packages *!pip install* is used to install those packages. For example, to install NumPy, run the code as displayed in 13 To install any library, the code will

```
!pip install numpy

Requirement already satisfied: numpy in c:\users\ashwi\anaconda3\lib\site-packages (1.19.5)
```

Figure 3: !pip install code

also be available in the following url (just enter the package name) - `https://pypi.org/project/`

5 Programming Environment Setup

The Jupyter Notebook is launched from the command prompt in order to start the execution environment for its implementation.Import all the libraries as displayed in Figure 4, Figure 5, Figure 6 and Figure 7

```python
#importing required packages
import numpy as np
import pandas as pd
import datetime as dt
pd.set_option('display.max_colwidth', None)          # To display all the data in each column
pd.options.display.max_columns = 50                 # To display every column of the dataset in head()
import warnings
warnings.filterwarnings('ignore')                   # To suppress all the warnings in the notebook.
import pandas_profiling as ppf
from datetime import timedelta
from numpy import mean
from numpy import std

#Packages to plot graph
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style='whitegrid', font_scale=1.3, color_codes=True)      # To apply seaborn styles to the plots.
import plotly.graph_objects as go
import squarify
import plotly.express as px
from matplotlib.gridspec import GridSpec
from pandas.plotting import scatter_matrix
```

Figure 4: Libraries for Preprocessing

```
#Importing Feature Selection Package
#from sklearn.feature_selection import RFE
#Importing K-Fold validation packages
from sklearn.model_selection import KFold
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from time import time

# Import required libraries
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import association_rules, apriori
from sklearn.metrics import f1_score
from sklearn.metrics import plot_roc_curve
from sklearn.model_selection import StratifiedKFold
```

Figure 5: Libraries for k-fold validation

```
#Classification ML Algorithms
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier

#Importing Classification Evaluation Metrics
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Figure 6: Libraries for ML models and Evaluation Metrics

```
# borderline-SMOTE for imbalanced dataset
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import BorderlineSMOTE
from matplotlib import pyplot
from imblearn.over_sampling import SMOTE
from  sklearn.metrics import accuracy_score
from numpy import where
# load and summarize the dataset
from pandas import read_csv
from collections import Counter
from matplotlib import pyplot
from sklearn.preprocessing import LabelEncoder
```

Figure 7: Libraries for SMOTE, Label encoder and Accuracy Score

5.1 Data Collection

The dataset utilized in this study is transactional data from a UK-based online retail gift shop named Online Retail II Data Source [1].The dataset was available in .csv format and was downloaded from Kaggle. The dataset had 8 columns and 1,067,371 records. The data was loaded as a DataFrame using python pandas library Figure 8.

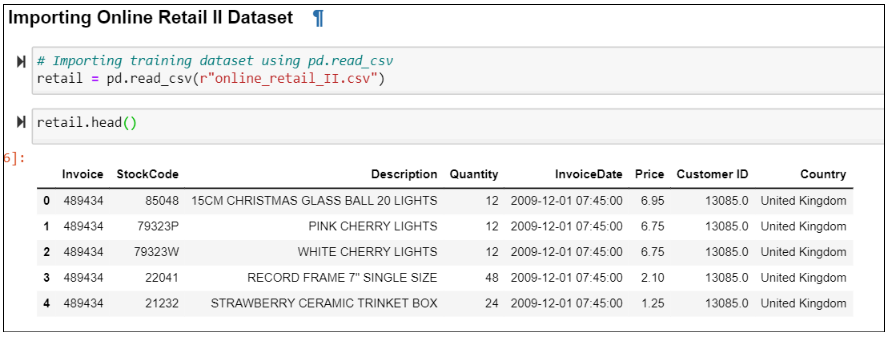


Figure 8: Loading Data to Pandas DataFrame

5.2 Execution of Code - Prerequisite

The Jupyter Notebook and the dataset should be uploaded to jupyter Notebook and should be placed in the same folder. **Important Note: Before executing the .ipynb file and the dataset should be placed in the same folder** Figure 10,

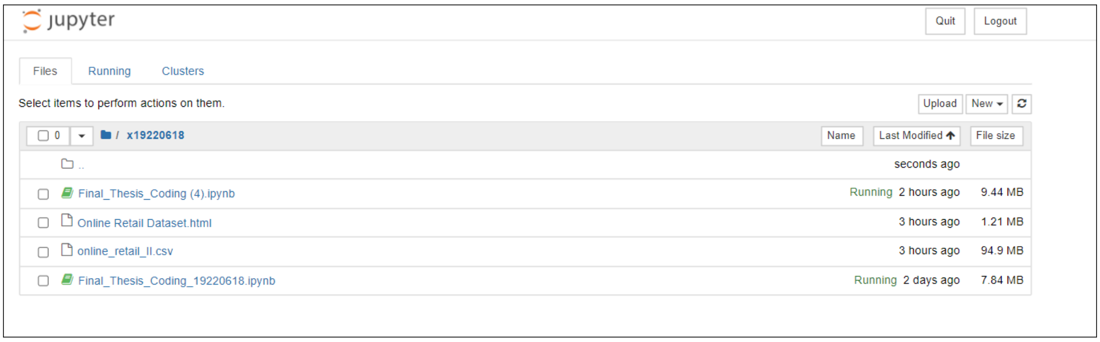


Figure 9: Code file and dataset to be loaded in Jupyter Notebook

Ensure that all the packages are installed and libraries are imported as mentioned in section 5.

Open the.ipynb file and go to Menu bar and click on 'Run All' to execute the entire file.

The progression of the code is explained with detailed screenshot in the below sections.

6 Data Pre-Processing

In this section, the data collection, pre-processing, feature creation performed on the dataset will be explained in terms of implementation.

[1]`https://www.kaggle.com/mashlyn/online-retail-ii-uci`

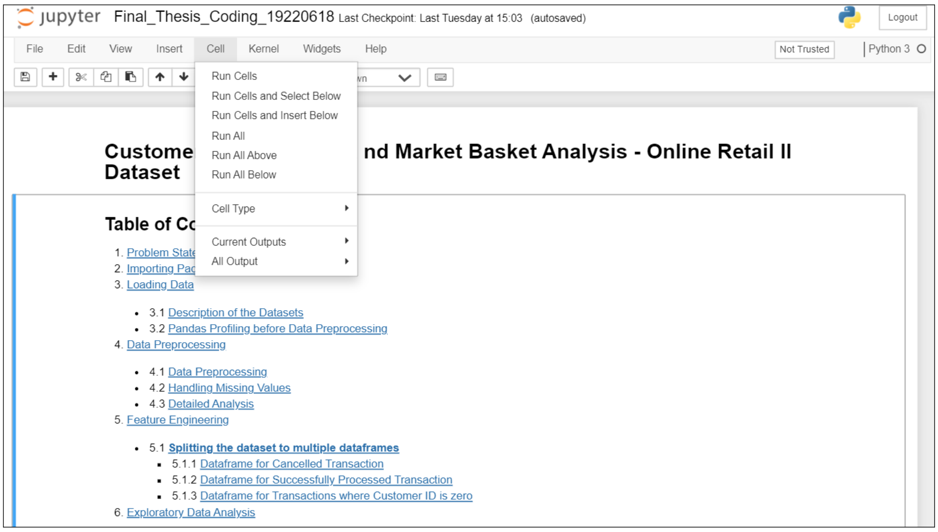


Figure 10: Execute All Cells

6.1 Pandas Profiling

As part of Data pre-processing, Pandas profiling was initially run to understand each attributes in depth Figure 11 and Figure 12

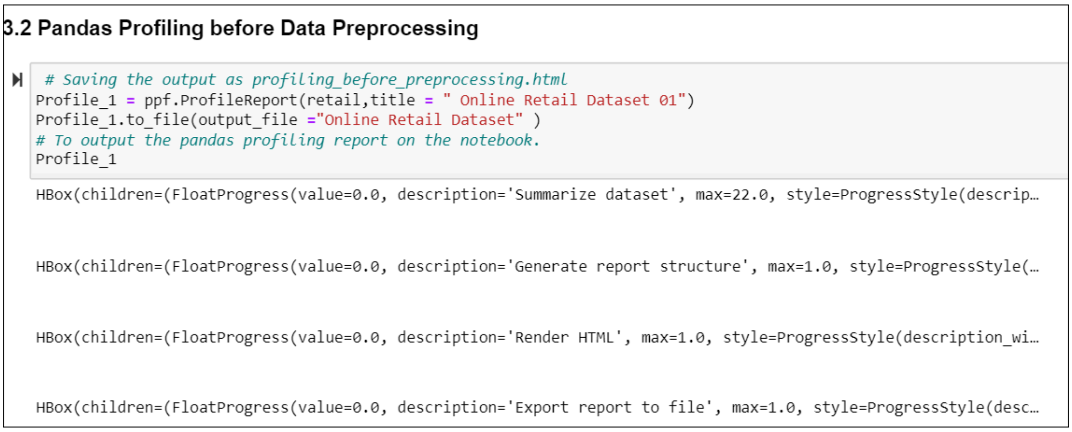


Figure 11: Pandas Profiling - Code

As highlighted in Figure 12, the variable, interactions, correlations, missing values, sample and duplicate rows all were explained in detail in the report. The duplicate records were deleted, and the missing values were replaced with a value that was not present in the database. For e.g., missing Customer id were replaced with '0' as no such value was present in the Customer id, and replacing with 0 helped understand the data better.

6.2 Exploratory Data Analysis

In this section, bar graphs, line plot, dashboards, pie plots, etc were plotted to understand and get useful insights from the dataset Figure 13 , Figure 14 and Figure 15.

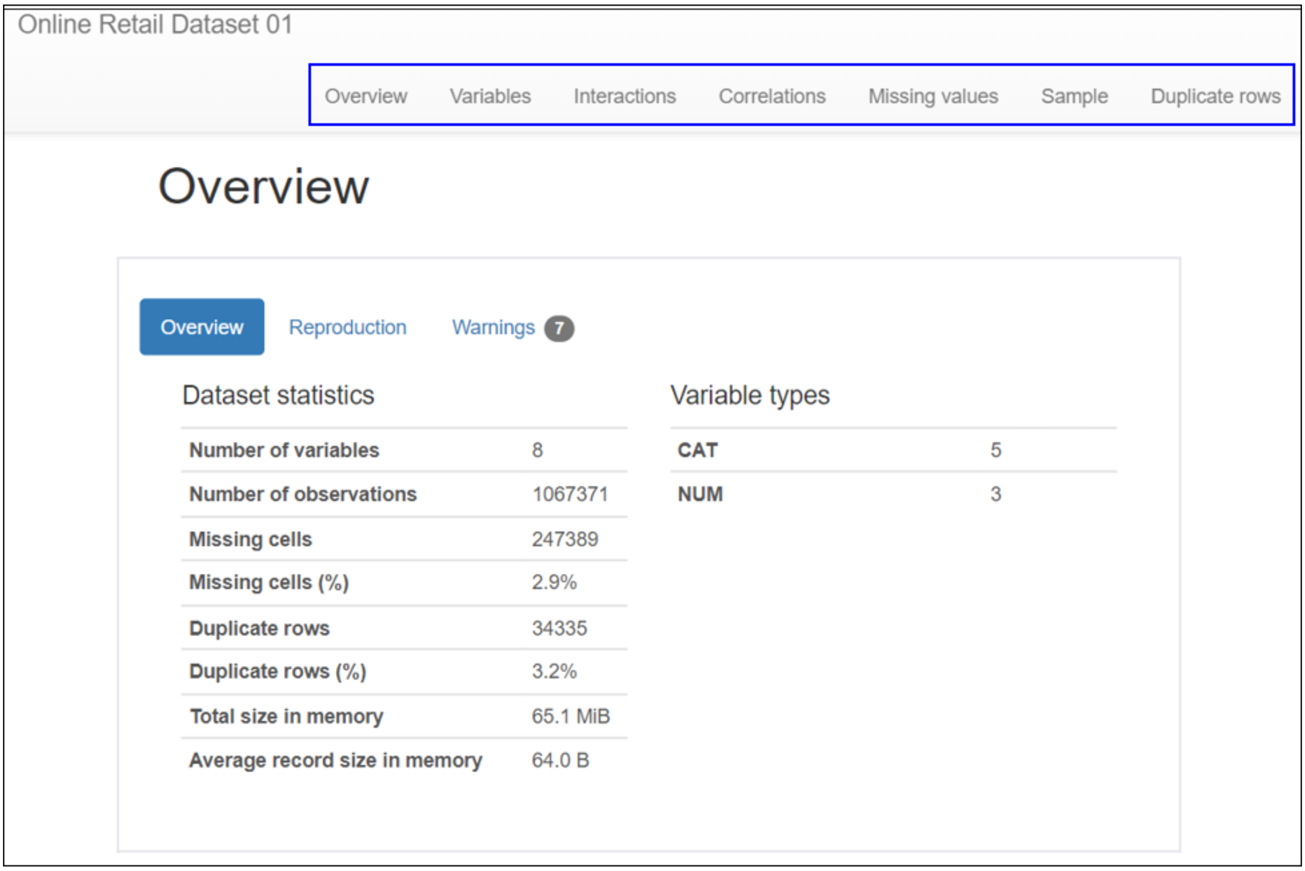
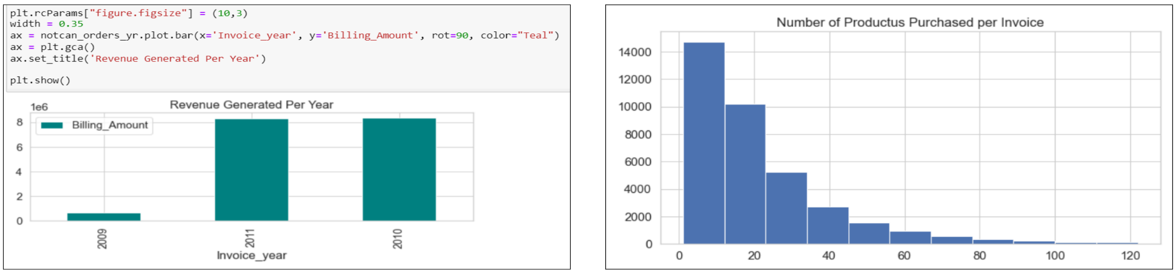
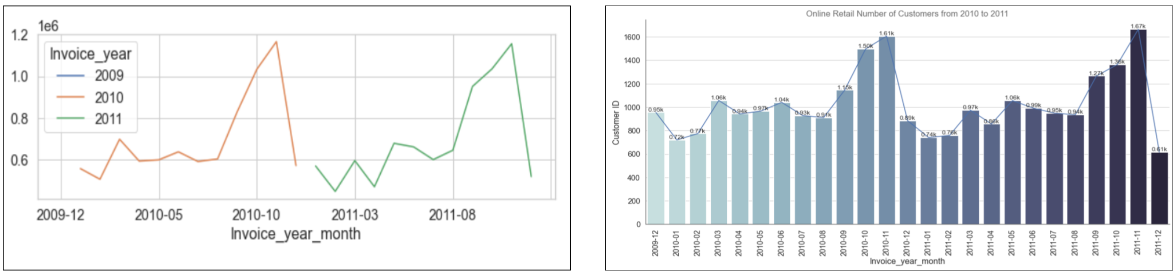


Figure 12: Pandas Profiling - Report



(a) Revenue Generated per year (b) Number of Products purchased per invoice

Figure 13: Revenue Generated per year and Number of Products per Invoice details



(a) Invoice trends for year 2010 and 2011 (b) Customer Count per month

Figure 14: Invoice Trends each year and Client Count per month

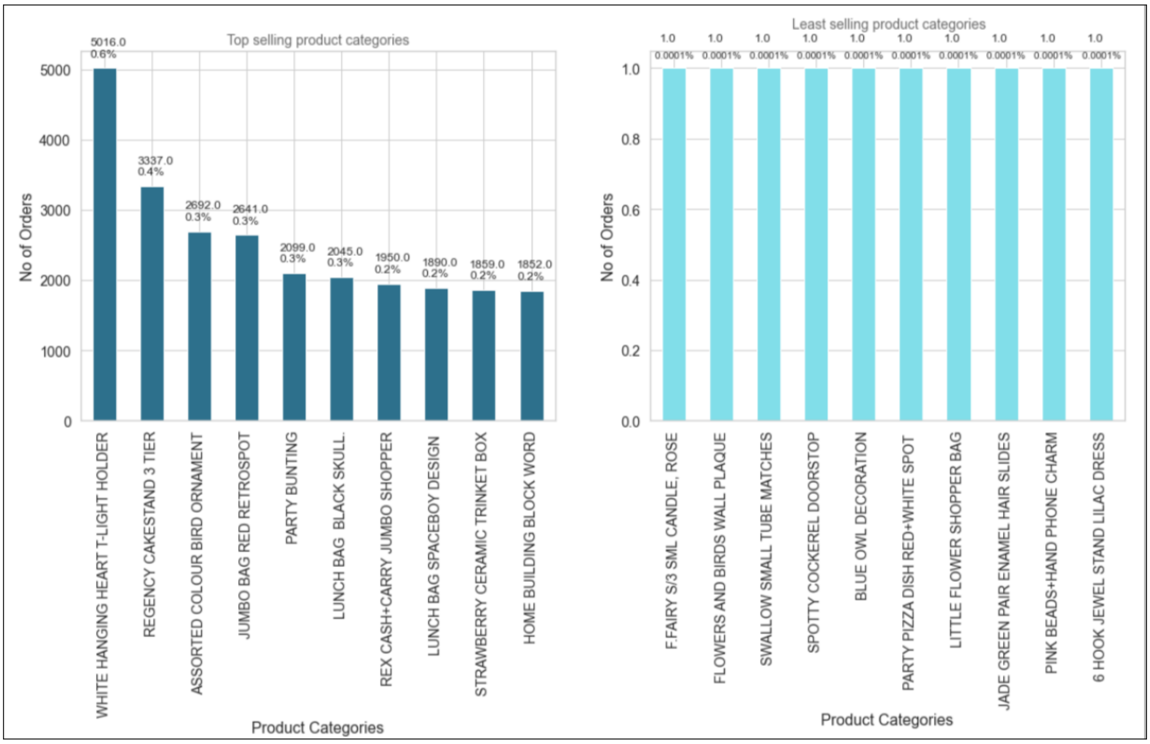


Figure 15: Top Selling Products and Least Selling Products

Based on the insights received, it was identified that only successful transaction will be used for implementation and a new dataframe for successful transaction was created Figure 16.

```
# Creating a new dataframe for the Successful transactions
notcan_orders = retail[~retail.isin(cancelled_orders)]
```

```
notcan_orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1033036 entries, 0 to 1067370
Data columns (total 15 columns):
 #   Column               Non-Null Count    Dtype
---  ------               --------------    -----
 0   Invoice              1013933 non-null  object
 1   StockCode            1013933 non-null  object
 2   Description          1013933 non-null  object
 3   Quantity             1013933 non-null  float64
 4   InvoiceDate          1013933 non-null  datetime64[ns]
 5   Price                1013933 non-null  float64
 6   Customer ID          1013933 non-null  float64
 7   Country              1013933 non-null  object
 8   Billing_Amount       1013933 non-null  float64
 9   Invoice_year         1013933 non-null  object
 10  Invoice_year_month1  1013933 non-null  object
 11  Invoice_year_month   1013933 non-null  object
 12  Invoice_year_day     1013933 non-null  object
 13  year_month_day       1013933 non-null  object
 14  time                 1013933 non-null  object
dtypes: datetime64[ns](1), float64(4), object(10)
memory usage: 126.1+ MB
```

Figure 16: Successful Transaction

7 Project Implementation

This section is divided into 3 parts: 1. Customer Segmentation 2. Multiclass Classification Modelling 3. Market Basket Analysis

7.1 Customer Segmentation using RFM and K-Means Clustering Technique

To perform segmentation, the Recency, Frequency and Monetary value of each consumer is calculated as shown in Figure 17 The RFM features extracted was then divided into

```
In [107]:  # --Group data by customerID--

           # Create snapshot date
           snapshot_date = notcan_orders['InvoiceDate'].max() + timedelta(days=1)
           print(snapshot_date)
           # Grouping by CustomerID
           data_process = notcan_orders.groupby(['Customer ID']).agg({
                   'InvoiceDate': lambda x: (snapshot_date - x.max()).days,
                   'Invoice': 'count',
                   'Billing_Amount': 'sum'}).reset_index()
           # Rename the columns
           data_process.rename(columns={'InvoiceDate': 'Recency',
                                 'Invoice': 'Frequency',
                              'Billing_Amount': 'MonetaryValue'}, inplace=True)

           2011-12-10 12:50:00
```

After getting the RFM values, a common practice is to create 'quartiles' on each of the metrics and assigning the required order. For example, suppose that we divide each metric into 4 cuts. For the recency metric, the highest value, 4, will be assigned to the customers with the least recency value (since they are the most recent customers). For the frequency and monetary metric, the highest value, 4, will be assigned to the customers with the Top 25% frequency and monetary values, respectively. After dividing the metrics into quartiles, we can collate the metrics into a single column (like a string of characters {like '213'}) to create classes of RFM values for our customers. We can divide the RFM metrics into lesser or more cuts depending on our requirements.

Figure 17: RFM Feature Creation

4 quintiles of 25% each, and assign a score of 1 to 4 to each Recency, Frequency and Monetary respectively. 1 is the highest value, and 4 is the lowest value. A final RFM score (Overall Value) is calculated simply by combining individual RFM score numbers as displayed in Figure 18

```
quantiles = data_process.quantile(q=[0.25,0.50,0.75])
quantiles = quantiles.to_dict()
```

```
# create two functions, to calculate the Quantile scores for Recency, Frequency and Monetary.
#RFM Score
    #1 - Potential
    #2 - Promising
    #3 - Can't Lose Them
    #4 - At Risk
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][0.5]:
        return 3
    elif x<=d[p][0.75]:
        return 2
    else:
        return 1
```

Figure 18: code to split data into quantiles

The RFM scale was then added up to get an RFM score Figure 19.

K-Means Clustering is then applied on the extracted feature to decide optimal number of segments appropriate for this dataset. However, k-means is effective when the data

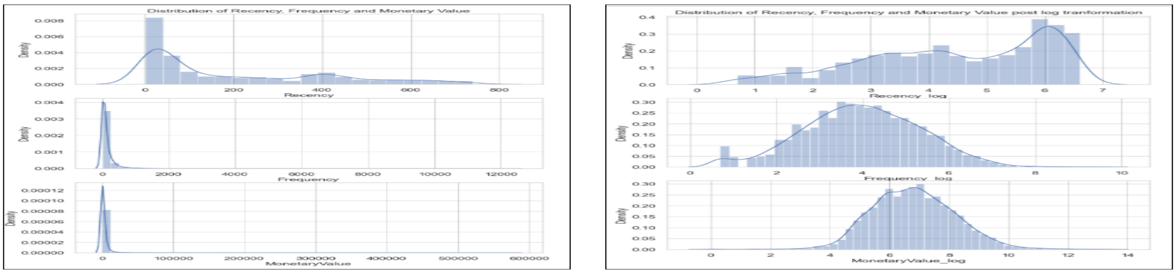
```
In [114]:  #concatenate the three score columns
           data_process['RFM_Segment'] = data_process.R_quartile.map(str)+data_process.F_quartile.map(str)+data_process.M_quartile.map(s
           data_process['RFM_Score'] = data_process[['R_quartile','F_quartile','M_quartile']].sum(axis=1)

In [115]:  data_process.head()

Out[115]:
```

	Customer ID	Recency	Frequency	MonetaryValue	R_quartile	F_quartile	M_quartile	RFM_Segment	RFM_Score
0	12346.0	326	34	77556.46	3	2	4	324	9
1	12347.0	2	222	4921.53	1	4	4	144	9
2	12348.0	75	51	2019.40	2	2	3	223	7
3	12349.0	19	175	4428.69	1	4	4	144	9
4	12350.0	310	17	334.40	3	1	1	311	5

RFM segmentation readily shows customer for any business like Best Customers, Loyal Customer, Customers on the verge of losing, Highest revenue-generating customers etc.

Figure 19: DataFrame with RFM Score

is distributed normally. Initially, the data was skewed and log function was applied to normalise the data Figure 20.



(a) Distribution of variables before data normalization

(b) Distribution of variables after data normalization

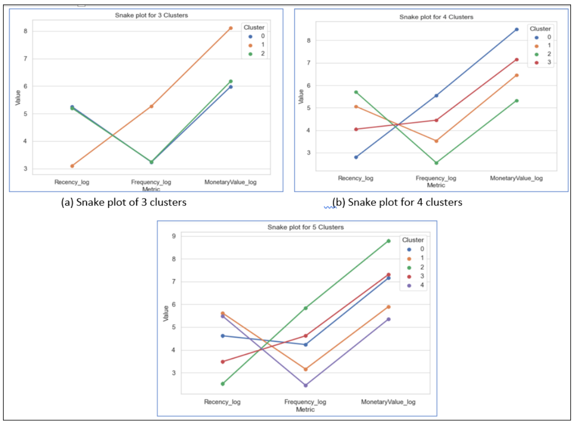
Figure 20: Distribution of variables pre and post data normalization

Elbow method and three-dimensional graph plot and snake plot were used to identify optimal clusters. To run flattened graph 'TSNE' package should be imported as shown in Figure 23 The flattened graph was bit confusing to decide the clusters Figure 22

```
from sklearn.manifold import TSNE

def kmeans(normalised_df_rfm, clusters_number, original_df_rfm):

    kmeans = KMeans(n_clusters = clusters_number, random_state = 1)
    kmeans.fit(normalised_df_rfm)
# Extract cluster labels
    cluster_labels = kmeans.labels_

    # Create a cluster label column in original dataset
    df_new = original_df_rfm.assign(Cluster = cluster_labels)

    # Initialise TSNE
    model = TSNE(random_state=1)
    transformed = model.fit_transform(df_new)

    # Plot t-SNE
    plt.title('Flattened Graph of {} Clusters'.format(clusters_number))
    sns.scatterplot(x=transformed[:,0], y=transformed[:,1], hue=cluster_labels, style=cluster_labels, palette="Set1")

    return df_new
plt.figure(figsize=(10, 10))
plt.subplot(3, 1, 1)
df_rfm_k3 = kmeans(RFM_Table_scaled, 3, RFM_table)
plt.subplot(3, 1, 2)
df_rfm_k4 = kmeans(RFM_Table_scaled, 4, RFM_table)
plt.subplot(3, 1, 3)
df_rfm_k5 = kmeans(RFM_Table_scaled, 5, RFM_table)
plt.tight_layout()
```

Figure 21: Importing Package TSNE and code for flattened graph

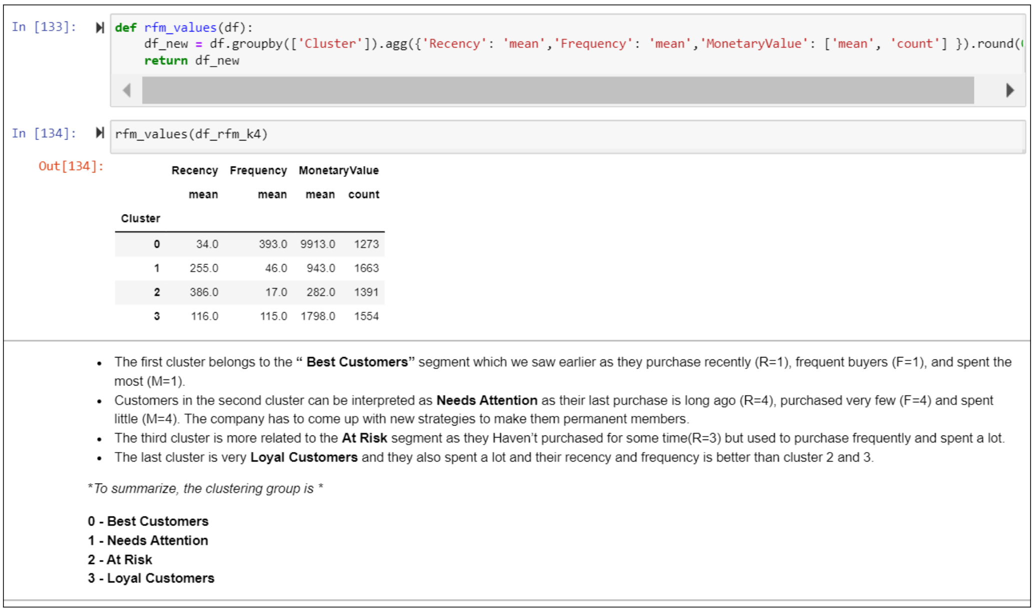
(a) Flattened graph for varied clusters



(b) Flattened graph for varied clusters

Figure 22: Cluster Selection based on flattened graph and Snake plot

Post analysis of snake plot, 4 clusters was considered as an optimal number of segments to split the data Figure 23



In [133]:
```
def rfm_values(df):
    df_new = df.groupby(['Cluster']).agg({'Recency': 'mean','Frequency': 'mean','MonetaryValue': ['mean', 'count'] }).round(
    return df_new
```

In [134]:
```
rfm_values(df_rfm_k4)
```

Out[134]:

	Recency	Frequency	MonetaryValue	
	mean	mean	mean	count
Cluster				
0	34.0	393.0	9913.0	1273
1	255.0	46.0	943.0	1663
2	386.0	17.0	282.0	1391
3	116.0	115.0	1798.0	1554

- The first cluster belongs to the " **Best Customers**" segment which we saw earlier as they purchase recently (R=1), frequent buyers (F=1), and spent the most (M=1).
- Customers in the second cluster can be interpreted as **Needs Attention** as their last purchase is long ago (R=4), purchased very few (F=4) and spent little (M=4). The company has to come up with new strategies to make them permanent members.
- The third cluster is more related to the **At Risk** segment as they Haven't purchased for some time(R=3) but used to purchase frequently and spent a lot.
- The last cluster is very **Loyal Customers** and they also spent a lot and their recency and frequency is better than cluster 2 and 3.

**To summarize, the clustering group is **

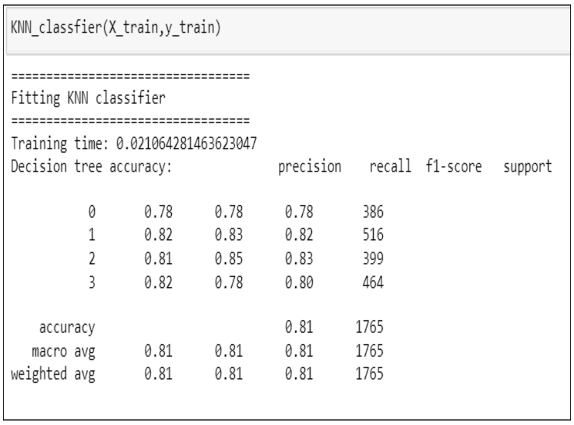
0 - Best Customers
1 - Needs Attention
2 - At Risk
3 - Loyal Customers

Figure 23: Customer Segments

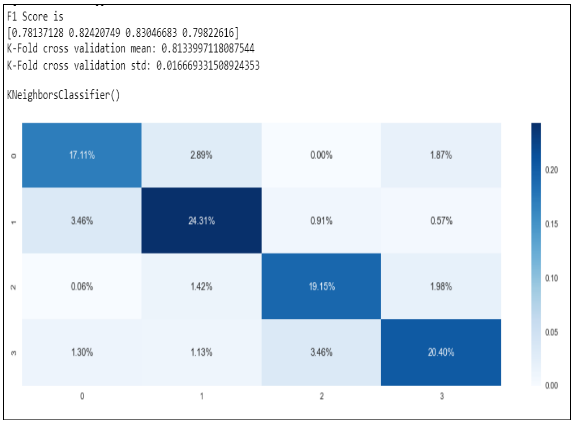
7.2 Multiclass - Classification Modelling

Before application Of models on data, the segments created are merged with each customer the imbalance is data clusters is verified, and then the data is split into train-test stratified split. Four models were applied - KNN Classifier, Random Forest Classifier, LGBM Classifier and Decision Tree Classifier

1. **KNN Classifier:**
   - Train Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 24.



```
KNN_classfier(X_train,y_train)

==================================
Fitting KNN classifier
==================================
Training time: 0.021064281463623047
Decision tree accuracy:              precision    recall  f1-score   support

           0       0.78      0.78      0.78       386
           1       0.82      0.83      0.82       516
           2       0.81      0.85      0.83       399
           3       0.82      0.78      0.80       464

    accuracy                           0.81      1765
   macro avg       0.81      0.81      0.81      1765
weighted avg       0.81      0.81      0.81      1765
```

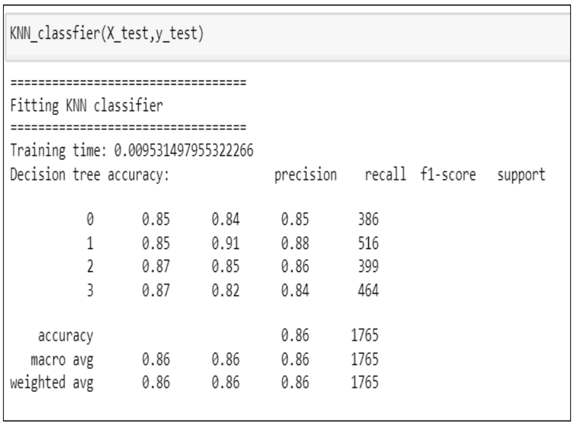


(a) Accuracy and Training time of KNN on train data

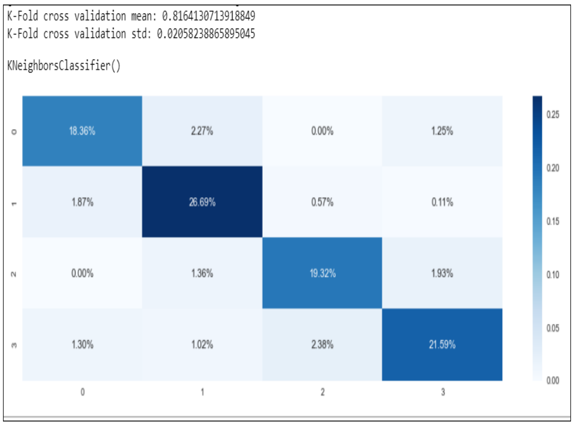
(b) Confusion Matrix and K-fold score of KNN on train data

Figure 24: Evaluation Matrix of KNN on Train Data

- Test Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 25.



```
KNN_classfier(X_test,y_test)

==================================
Fitting KNN classifier
==================================
Training time: 0.009531497955322266
Decision tree accuracy:              precision    recall  f1-score   support

           0       0.85      0.84      0.85       386
           1       0.85      0.91      0.88       516
           2       0.87      0.85      0.86       399
           3       0.87      0.82      0.84       464

    accuracy                           0.86      1765
   macro avg       0.86      0.86      0.86      1765
weighted avg       0.86      0.86      0.86      1765
```



(a) Accuracy and Training time of KNN on test data

(b) Confusion Matrix and K-fold score of KNN on test data

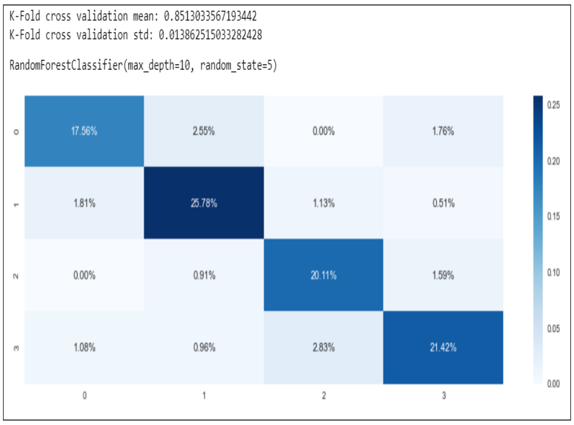
Figure 25: Evaluation Matrix of KNN on Test Data

2. **Random Forest Classifier:**

- Train Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 26 and Figure 27.

```
random_forest_classfier(X_train,y_train)

==================================
Fitting RandomForest classifier
==================================
Training time: 1.0322661399841309
Decision tree accuracy:              precision    recall  f1-score   support

           0       0.86      0.80      0.83       386
           1       0.85      0.88      0.87       516
           2       0.84      0.89      0.86       399
           3       0.85      0.81      0.83       464

    accuracy                           0.85      1765
   macro avg       0.85      0.85      0.85      1765
weighted avg       0.85      0.85      0.85      1765
```

(a) Accuracy and Training time of RFC on train data



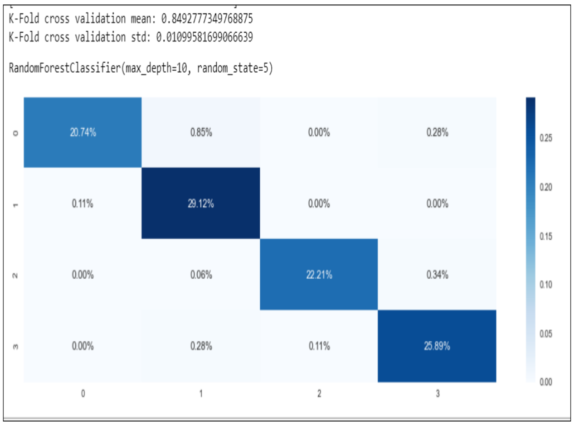
(b) Confusion Matrix and K-fold score of RFC on train data

Figure 26: Evaluation Matrix of RFC on Train Data

- Test Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 27.

```
random_forest_classfier(X_test,y_test)

==================================
Fitting RandomForest classifier
==================================
Training time: 0.5812742710113525
Decision tree accuracy:              precision    recall  f1-score   support

           0       0.99      0.95      0.97       386
           1       0.96      1.00      0.98       516
           2       0.99      0.98      0.99       399
           3       0.98      0.98      0.98       464

    accuracy                           0.98      1765
   macro avg       0.98      0.98      0.98      1765
weighted avg       0.98      0.98      0.98      1765
```

(a) Accuracy and Training time of RFC on test data

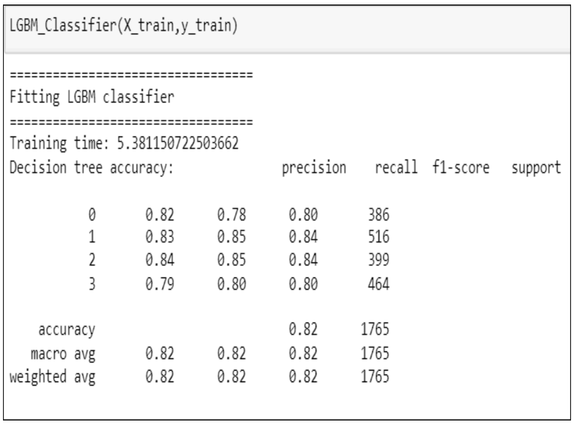


(b) Confusion Matrix and K-fold score of RFC on test data

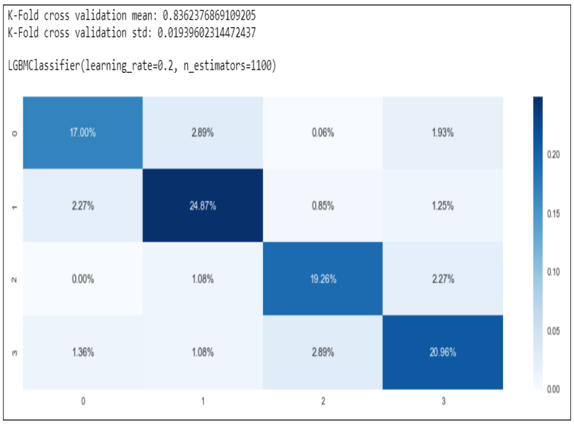
Figure 27: Evaluation Matrix of RFC on Test Data

3. **LGBM:**

   - Train Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 28 and Figure 29.



```
LGBM_Classifier(X_train,y_train)

=================================
Fitting LGBM classifier
=================================
Training time: 5.381150722503662
Decision tree accuracy:            precision    recall  f1-score   support

           0       0.82      0.78      0.80       386
           1       0.83      0.85      0.84       516
           2       0.84      0.85      0.84       399
           3       0.79      0.80      0.80       464

    accuracy                           0.82      1765
   macro avg       0.82      0.82      0.82      1765
weighted avg       0.82      0.82      0.82      1765
```

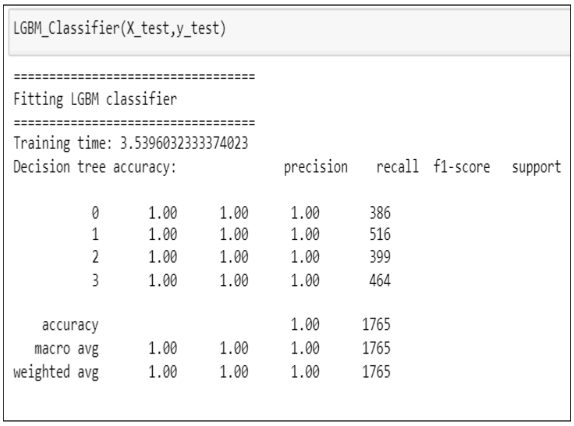


(a) Accuracy and Training time of LGBM on train data

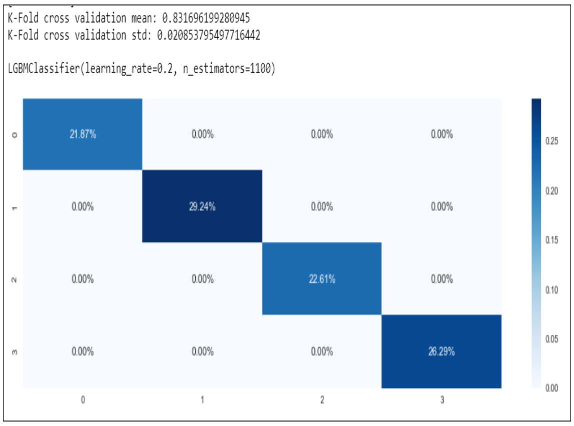
(b) Confusion Matrix and K-fold score of LGBM on train data

Figure 28: Evaluation Matrix of LGBM on Train Data

- Test Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 29.



```
LGBM_Classifier(X_test,y_test)

=================================
Fitting LGBM classifier
=================================
Training time: 3.5396032333374023
Decision tree accuracy:            precision    recall  f1-score   support

           0       1.00      1.00      1.00       386
           1       1.00      1.00      1.00       516
           2       1.00      1.00      1.00       399
           3       1.00      1.00      1.00       464

    accuracy                           1.00      1765
   macro avg       1.00      1.00      1.00      1765
weighted avg       1.00      1.00      1.00      1765
```



(a) Accuracy and Training time of LGBM on test data

(b) Confusion Matrix and K-fold score of LGBM on test data

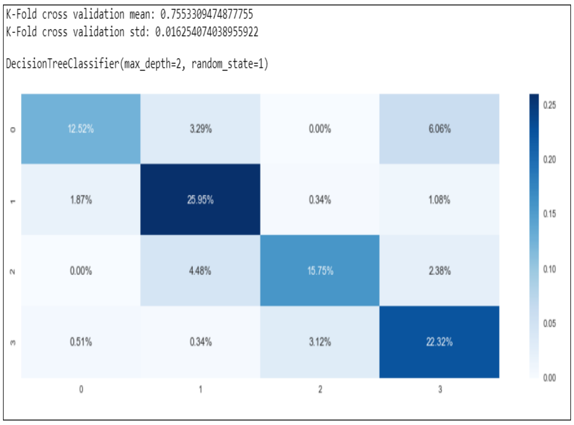
Figure 29: Evaluation Matrix of LGBM on Test Data

4. **Decision Tree Classifier:**

- Train Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 30 and Figure 31.

```
decision_tree_classifier(X_train,y_train)

==================================
Decision classfier classifier
==================================
Training time: 0.02495431900024414
Decision tree accuracy:                precision    recall  f1-score   support

           0       0.84      0.57      0.68       386
           1       0.76      0.89      0.82       516
           2       0.82      0.70      0.75       399
           3       0.70      0.85      0.77       464

    accuracy                           0.77      1765
   macro avg       0.78      0.75      0.76      1765
weighted avg       0.78      0.77      0.76      1765
```

(a) Accuracy and Training time of DTC on train data



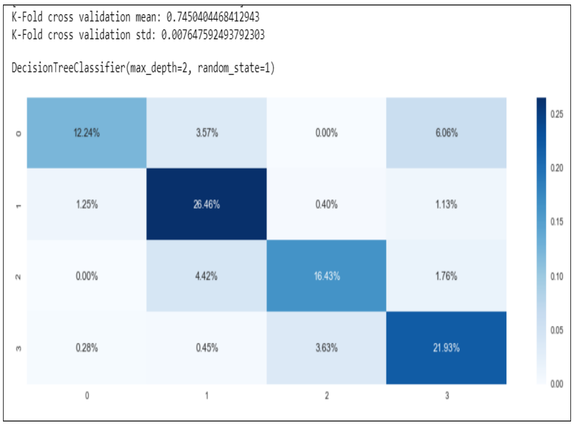
(b) Confusion Matrix and K-fold score of DTC on train data

Figure 30: Evaluation Matrix of RFC on Train Data

- Test Data Evaluation: Evaluation was based on training time, accuracy and confusion matrix Figure 31.

```
decision_tree_classifier(X_test,y_test)

==================================
Decision classfier classifier
==================================
Training time: 0.007982969284057617
Decision tree accuracy:                precision    recall  f1-score   support

           0       0.89      0.56      0.69       386
           1       0.76      0.91      0.83       516
           2       0.80      0.73      0.76       399
           3       0.71      0.83      0.77       464

    accuracy                           0.77      1765
   macro avg       0.79      0.76      0.76      1765
weighted avg       0.78      0.77      0.77      1765
```

(a) Accuracy and Training time of DTC on test data



(b) Confusion Matrix and K-fold score of DTC on test data

Figure 31: Evaluation Matrix of DTC on Test Data

7.3 Market Basket Analysis

To implement market basket analysis at segment level, the data was grouped based on the cluster as seen in Figure 32. The recommended products for each cluster are depicted in Figure 34, Figure 35, Figure 36 and Figure 10

```python
#Split the dataset into to  different dataframes based on their clusters

# **0 - Best Customers<br>**
# **1 - Needs Attention<br>**
# **2 - At Risk<br>**
# **3 - Loyal Customers<br>**
#create unique list of names
UniqueNames1 = final_mba.Cluster.unique()

#create a data frame dictionary to store your data frames
DataFrameDict2 = {elem : pd.DataFrame for elem in UniqueNames1}

for key in DataFrameDict2.keys():
    DataFrameDict2[key] = final_mba[:][final_mba.Cluster == key]
```

Figure 32: Recommended product for Best Customers

1. **Association Mining Rule on 'Best Customer Cluster'**

```python
frequent_itemsets_plus[ (frequent_itemsets_plus['length']  >= 2) &
                        (frequent_itemsets_plus['support'] >= 0.03) ]
```

	support	itemsets	length
66	0.037740	(WOODEN FRAME ANTIQUE WHITE , WOODEN PICTURE FRAME WHITE FINISH)	2
77	0.036327	(WHITE HANGING HEART T-LIGHT HOLDER, RED HANGING HEART T-LIGHT HOLDER)	2

```python
association_rules(frequent_itemsets_plus, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(WOODEN FRAME ANTIQUE WHITE)	(WOODEN PICTURE FRAME WHITE FINISH)	0.063623	0.060293	0.037740	0.593180	9.838350	0.033904	2.309885
1	(WOODEN PICTURE FRAME WHITE FINISH)	(WOODEN FRAME ANTIQUE WHITE)	0.060293	0.063623	0.037740	0.625941	9.838350	0.033904	2.503291
2	(WHITE HANGING HEART T-LIGHT HOLDER)	(RED HANGING HEART T-LIGHT HOLDER)	0.146367	0.051261	0.036327	0.248190	4.841665	0.028824	1.261940
3	(RED HANGING HEART T-LIGHT HOLDER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.051261	0.146367	0.036327	0.708661	4.841665	0.028824	2.930037

Figure 33: Recommended product for 'Best Customers' segment

2. **Association Mining Rule on 'Needs Attention' Customer Cluster**

```
frequent_itemsets_plus1[ (frequent_itemsets_plus1['length']  >= 2) &
                         (frequent_itemsets_plus1['support'] >= 0.03) ]
```

4]:

support	itemsets	length

```
association_rules(frequent_itemsets_plus1, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

5]:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
frequent_itemsets_plus1[ (frequent_itemsets_plus1['length']  >= 2) &
                         (frequent_itemsets_plus1['support'] >= 0.01) ]
```

1]:

support	itemsets	length

```
association_rules(frequent_itemsets_plus1, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

2]:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

Figure 34: Recommended product for 'Needs Attention' customer segment

3. **Association Mining Rule on 'At Risks' Customer Cluster**

```
frequent_itemsets_plus2[ (frequent_itemsets_plus2['length']  >= 2) &
                         (frequent_itemsets_plus2['support'] >= 0.03) ]
```

]:

support	itemsets	length

```
b = frequent_itemsets_plus2[frequent_itemsets_plus2['length'] >1]
b
```

]:

support	itemsets	length

```
association_rules(frequent_itemsets_plus2, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

]:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

```
frequent_itemsets_plus2[ (frequent_itemsets_plus2['length']  >= 2) &
                         (frequent_itemsets_plus2['support'] >= 0.01) ]
```

]:

support	itemsets	length

```
association_rules(frequent_itemsets_plus2, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

]:

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction

Figure 35: Recommended product for 'At Risk' Customer Segment

4. **Applying Association Mining Rule on At Loyal Customer Cluster**

```
frequent_itemsets_plus3[ (frequent_itemsets_plus3['length']  >= 2) &
                         (frequent_itemsets_plus3['support'] >= 0.01) ]
```

	support	itemsets	length
40	0.034027	(WHITE HANGING HEART T-LIGHT HOLDER, RED HANGING HEART T-LIGHT HOLDER)	2

```
association_rules(frequent_itemsets_plus3, metric='lift',
                  min_threshold=1).sort_values('lift', ascending=False).reset_index(drop=True)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(RED HANGING HEART T-LIGHT HOLDER)	(WHITE HANGING HEART T-LIGHT HOLDER)	0.047736	0.150306	0.034027	0.712821	4.742462	0.026852	2.958756
1	(WHITE HANGING HEART T-LIGHT HOLDER)	(RED HANGING HEART T-LIGHT HOLDER)	0.150306	0.047736	0.034027	0.226384	4.742462	0.026852	1.230927

Figure 36: Recommended product for 'Loyal' Customer segment