CDS503_G7

December 16, 2019

1 Table of Contents

2 Explanatory Data Analysis (EDA)

The purpose of this part is to explore, understand and clean the data

2.1 Input Data

[1]:
```
# Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import datetime as dt
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
#%pylab inline
from IPython.display import display, Image
import os
import pydotplus

import datetime, nltk, warnings
#nltk.download()

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn import svm
from sklearn.svm import SVC
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.model_selection import cross_val_score
```

```
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\externals\six.py:31: DeprecationWarning: The module is
deprecated in version 0.21 and will be removed in version 0.23 since we've
dropped support for Python 2.7. Please rely on the official version of six
(https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

Due to the given data is considered as unregistered file, hence need to utilize sublime text 3 to solve this problem. After the data is saved in csv.file, the file will be transfer to sublime text 3 to save with encoding in "utf-8". Hence, the dataset can be apply in python more smoothly.

[2]:
```
data = pd.read_csv('C:/Users/tiliew/Documents/TianChin Specific/MasterClass/
↪CDS503-ML/Project/online_retail_II_0910.csv', encoding = "utf-8")
#print(data.describe(include='all'))
print(data.info())
```

```
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525461 entries, 0 to 525460
Data columns (total 8 columns):
Invoice         525461 non-null object
StockCode       525461 non-null object
Description     522533 non-null object
Quantity        525461 non-null int64
InvoiceDate     525461 non-null object
Price           525461 non-null float64
Customer ID     417534 non-null float64
Country         525461 non-null object
dtypes: float64(2), int64(1), object(5)
memory usage: 32.1+ MB
None
```

[2]:

```
  Invoice StockCode                          Description  Quantity  \
0  489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1  489434    79323P                   PINK CHERRY LIGHTS        12
2  489434    79323W                  WHITE CHERRY LIGHTS        12
3  489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4  489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24

      InvoiceDate  Price  Customer ID         Country
0  12/1/2009 7:45   6.95      13085.0  United Kingdom
1  12/1/2009 7:45   6.75      13085.0  United Kingdom
2  12/1/2009 7:45   6.75      13085.0  United Kingdom
3  12/1/2009 7:45   2.10      13085.0  United Kingdom
4  12/1/2009 7:45   1.25      13085.0  United Kingdom
```

There are two attributes have different range index compared to other attributes which are description and customer ID. Let's find the null value in those attributes.

[3]:

```
data.isnull().sum()
```

[3]:

```
Invoice             0
StockCode           0
Description      2928
Quantity            0
InvoiceDate         0
Price               0
Customer ID    107927
Country             0
dtype: int64
```

2.2 Data Cleaning

Two attributes which are description and customer ID consist of missing values. Hence we will choose to remove missing value or null values from the dataset instead of do replacement with mean or median. After remove the missing values from two related attributes, the range index is already reduce to 406829 entries instead of 541908 entries.

[4]:
```
data['Description'].replace(0, np.nan, inplace= True)
data['Customer ID'].replace(0, np.nan, inplace= True)

# Remove Missing Values from Related Attributes
df = data.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 417534 entries, 0 to 525460
Data columns (total 8 columns):
Invoice        417534 non-null object
StockCode      417534 non-null object
Description    417534 non-null object
Quantity       417534 non-null int64
InvoiceDate    417534 non-null object
Price          417534 non-null float64
Customer ID    417534 non-null float64
Country        417534 non-null object
dtypes: float64(2), int64(1), object(5)
memory usage: 28.7+ MB
```

Check again to prove that there are no any missing values after remove unwanted null values.

[5]:
```
df.isnull().sum()
```

[5]:
```
Invoice        0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
Price          0
Customer ID    0
Country        0
dtype: int64
```

[6]:
```
df.describe(include='all')
```

[6]:
```
       Invoice StockCode                         Description       Quantity  \
count   417534    417534                              417534  417534.000000
unique   23587      4031                                4459            NaN
top     500356    85123A  WHITE HANGING HEART T-LIGHT HOLDER            NaN
freq       270      3245                                3245            NaN
```

```
mean       NaN      NaN                                  NaN     12.758815
std        NaN      NaN                                  NaN    101.220424
min        NaN      NaN                                  NaN  -9360.000000
25%        NaN      NaN                                  NaN      2.000000
50%        NaN      NaN                                  NaN      4.000000
75%        NaN      NaN                                  NaN     12.000000
max        NaN      NaN                                  NaN  19152.000000

           InvoiceDate          Price    Customer ID         Country
count           417534  417534.000000  417534.000000          417534
unique           21786            NaN            NaN              37
top     3/7/2010 15:34            NaN            NaN  United Kingdom
freq               270            NaN            NaN          379423
mean               NaN       3.887547   15360.645478             NaN
std                NaN      71.131797    1680.811316             NaN
min                NaN       0.000000   12346.000000             NaN
25%                NaN       1.250000   13983.000000             NaN
50%                NaN       1.950000   15311.000000             NaN
75%                NaN       3.750000   16799.000000             NaN
max                NaN   25111.090000   18287.000000             NaN
```

After illustrate the description, found out the minimum value from “quantity” has negative value. In actual transaction, there is no such thing as negative quantities. Hence we decided tp require to discard the negative value from “quantity”. After remove the negative value from quantity to positive value, now the dataset consists of 397924 entries to proceed with following step.

[7]:
```python
# Extract the rows with quantity > 0
df[df.Quantity > 0]
```

[7]:
```
       Invoice StockCode                          Description  Quantity  \
0       489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1       489434    79323P                   PINK CHERRY LIGHTS        12
2       489434    79323W                  WHITE CHERRY LIGHTS        12
3       489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4       489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24
…          …       …                                     …        …
525456  538171     22271                    FELTCRAFT DOLL ROSIE         2
525457  538171     22750                FELTCRAFT PRINCESS LOLA DOLL     1
525458  538171     22751              FELTCRAFT PRINCESS OLIVIA DOLL     1
525459  538171     20970          PINK FLORAL FELTCRAFT SHOULDER BAG     2
525460  538171     21931                     JUMBO STORAGE BAG SUKI      2

           InvoiceDate  Price  Customer ID         Country
0       12/1/2009 7:45   6.95      13085.0  United Kingdom
1       12/1/2009 7:45   6.75      13085.0  United Kingdom
2       12/1/2009 7:45   6.75      13085.0  United Kingdom
3       12/1/2009 7:45   2.10      13085.0  United Kingdom
```

```
4         12/1/2009 7:45   1.25      13085.0  United Kingdom
…                    …    …          …            …
525456  12/9/2010 20:01   2.95      17530.0  United Kingdom
525457  12/9/2010 20:01   3.75      17530.0  United Kingdom
525458  12/9/2010 20:01   3.75      17530.0  United Kingdom
525459  12/9/2010 20:01   3.75      17530.0  United Kingdom
525460  12/9/2010 20:01   1.95      17530.0  United Kingdom

[407695 rows x 8 columns]
```

'United Kingdom' is only one selected country from dataset and the total amount of 'United Kingdom' from dataset are 354345. The reason why we go for United Kingdom is because it has the most transaction occur in that country. The rest of the countries are discarded.

[8]:
```
df_clean = df[df.Country=='United Kingdom']
df_clean
```

[8]:
```
       Invoice StockCode                          Description  Quantity  \
0       489434     85048  15CM CHRISTMAS GLASS BALL 20 LIGHTS        12
1       489434    79323P                   PINK CHERRY LIGHTS        12
2       489434    79323W                  WHITE CHERRY LIGHTS        12
3       489434     22041         RECORD FRAME 7" SINGLE SIZE         48
4       489434     21232       STRAWBERRY CERAMIC TRINKET BOX        24
…          …        …                                   …       …
525456  538171     22271                    FELTCRAFT DOLL ROSIE         2
525457  538171     22750             FELTCRAFT PRINCESS LOLA DOLL        1
525458  538171     22751           FELTCRAFT PRINCESS OLIVIA DOLL        1
525459  538171     20970    PINK FLORAL FELTCRAFT SHOULDER BAG         2
525460  538171     21931                  JUMBO STORAGE BAG SUKI         2

             InvoiceDate  Price  Customer ID         Country
0         12/1/2009 7:45   6.95      13085.0  United Kingdom
1         12/1/2009 7:45   6.75      13085.0  United Kingdom
2         12/1/2009 7:45   6.75      13085.0  United Kingdom
3         12/1/2009 7:45   2.10      13085.0  United Kingdom
4         12/1/2009 7:45   1.25      13085.0  United Kingdom
…                    …    …          …            …
525456  12/9/2010 20:01   2.95      17530.0  United Kingdom
525457  12/9/2010 20:01   3.75      17530.0  United Kingdom
525458  12/9/2010 20:01   3.75      17530.0  United Kingdom
525459  12/9/2010 20:01   3.75      17530.0  United Kingdom
525460  12/9/2010 20:01   1.95      17530.0  United Kingdom

[379423 rows x 8 columns]
```

3 Product Clustering

3.1 Products Description

As a first step, We extract from the Description variable the information that will prove useful. To do this, We use the following function:

[9]:
```python
is_noun = lambda pos: pos[:2] == 'NN'

def keywords_inventory(dataframe, column = 'Description'):
    stemmer = nltk.stem.SnowballStemmer("english")
    keywords_roots  = dict()  # collect the words / root
    keywords_select = dict()  # association: root <-> keyword
    category_keys   = []
    count_keywords  = dict()
    icount = 0
    for s in dataframe[column]:
        if pd.isnull(s): continue
        lines = s.lower()
        tokenized = nltk.word_tokenize(lines)
        nouns = [word for (word, pos) in nltk.pos_tag(tokenized) if␣
 ↪is_noun(pos)]

        for t in nouns:
            t = t.lower() ; racine = stemmer.stem(t)
            if racine in keywords_roots:
                keywords_roots[racine].add(t)
                count_keywords[racine] += 1
            else:
                keywords_roots[racine] = {t}
                count_keywords[racine] = 1

    for s in keywords_roots.keys():
        if len(keywords_roots[s]) > 1:
            min_length = 1000
            for k in keywords_roots[s]:
                if len(k) < min_length:
                    clef = k ; min_length = len(k)
            category_keys.append(clef)
            keywords_select[s] = clef
        else:
            category_keys.append(list(keywords_roots[s])[0])
            keywords_select[s] = list(keywords_roots[s])[0]

    print("Nb of keywords in variable '{}': {}".
 ↪format(colonne,len(category_keys)))
    return category_keys, keywords_roots, keywords_select, count_keywords
```

This function takes as input the dataframe and analyzes the content of the Description column by performing the following operations:

extract the names (proper, common) appearing in the products description for each name, I extract the root of the word and aggregate the set of names associated with this particular root count the number of times each root appears in the dataframe when several words are listed for the same root, I consider that the keyword associated with this root is the shortest name (this systematically selects the singular when there are singular/plural variants) The first step of the analysis is to retrieve the list of products:

[10]:
```
df_product = pd.DataFrame(df_clean['Description'].unique()).rename(columns = {0:
 ↪'Description'})
```

[11]:
```
df_product.dtypes
```

[11]:
```
Description    object
dtype: object
```

Once this list is created, I use the function I previously defined in order to analyze the description of the various products:

[12]:
```
keywords, keywords_roots, keywords_select, count_keywords =␣
 ↪keywords_inventory(df_product)
```

```
Nb of keywords in variable 'Description': 1570
```

The execution of this function returns three variables:

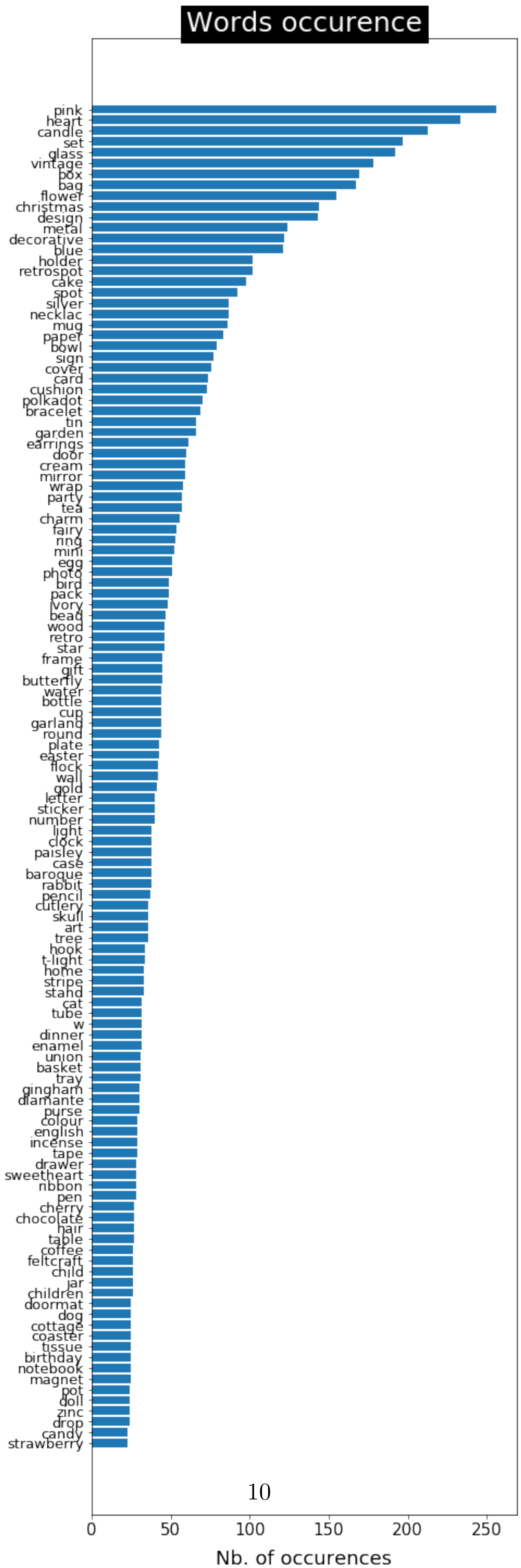
keywords: the list of extracted keywords keywords_roots: a dictionary where the keys are the keywords roots and the values are the lists of words associated with those roots count_keywords: dictionary listing the number of times every word is used At this point, I convert the count_keywords dictionary into a list, to sort the keywords according to their occurences:

[13]:
```
list_products = []
for k,v in count_keywords.items():
    list_products.append([keywords_select[k],v])
list_products.sort(key = lambda x:x[1], reverse = True)
```

Using it, I create a representation of the most common keywords:

[14]:
```
liste = sorted(list_products, key = lambda x:x[1], reverse = True)
#_______________________________
plt.rc('font', weight='normal')
fig, ax = plt.subplots(figsize=(7, 25))
y_axis = [i[1] for i in liste[:125]]
x_axis = [k for k,i in enumerate(liste[:125])]
x_label = [i[0] for i in liste[:125]]
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 13)
plt.yticks(x_axis, x_label)
```

```
plt.xlabel("Nb. of occurences", fontsize = 18, labelpad = 10)
ax.barh(x_axis, y_axis, align = 'center')
ax = plt.gca()
ax.invert_yaxis()
#---------------------------------------------------------------------------------------
plt.title("Words occurence",bbox={'facecolor':'k', 'pad':5}, color='w',fontsize = 25)
plt.show()
```


Words occurence
pink
heart
candle
set
glass
vintage
box
bag
flower
christmas
design
metal
decorative
blue
holder
retrospot
cake
spot
silver
necklac
mug
paper
bowl
sign
cover
card
cushion
polkadot
bracelet
tin
garden
earrings
door
cream
mirror
wrap
party
tea
charm
fairy
ring
mini
egg
photo
bird
pack
ivory
bead
wood
retro
star
frame
gift
butterfly
water
bottle
cup
garland
round
plate
easter
flock
wall
gold
letter
sticker
number
light
clock
paisley
case
baroque
rabbit
pencil
cutlery
skull
art
tree
hook
t-light
home
stripe
stand
cat
tube
w
dinner
enamel
union
basket
tray
gingham
diamante
purse
colour
english
incense
tape
drawer
sweetheart
ribbon
pen
cherry
chocolate
hair
table
coffee
feltcraft
child
jar
children
doormat
dog
cottage
coaster
tissue
birthday
notebook
magnet
pot
doll
zinc
drop
candy
strawberry
0
50
100
150
200
250
Nb. of occurences

3.2 Defining product categories

The list that was obtained contains more than 1500 keywords and the most frequent ones appear in more than 200 products. However, while examinating the content of the list, I note that some names are useless. Others are do not carry information, like colors. Therefore, I discard these words from the analysis that follows and also, I decide to consider only the words that appear more than 13 times.

[15]:
```
list_products = []
for k,v in count_keywords.items():
    word = keywords_select[k]
    if word in ['pink', 'blue', 'tag', 'green', 'orange']: continue
    if len(word) < 3 or v < 13: continue
    if ('+' in word) or ('/' in word): continue
    list_products.append([word, v])
#______________________________________________________
list_products.sort(key = lambda x:x[1], reverse = True)
print('Number of Distinct Products:', len(list_products))
```

```
mots conservés: 225
```

Now I will use these keywords to create groups of product. Firstly, I define the X matrix as:

	word 1	…..	word j	…..	word N
product 1	a1,1				a1,N
…			…		
product i	…		ai,j		…
…			…		
product M	a1,1				aM,N

where the ai,j coefficient is 1 if the description of the product i contains the word j , and 0 otherwise.

[16]:
```
list_products = df_clean['Description'].unique()
X = pd.DataFrame()
for key, occurence in list_products:
    X.loc[:, key] = list(map(lambda x:int(key.upper() in x), list_products))
```

3.3 K-Means Clustering

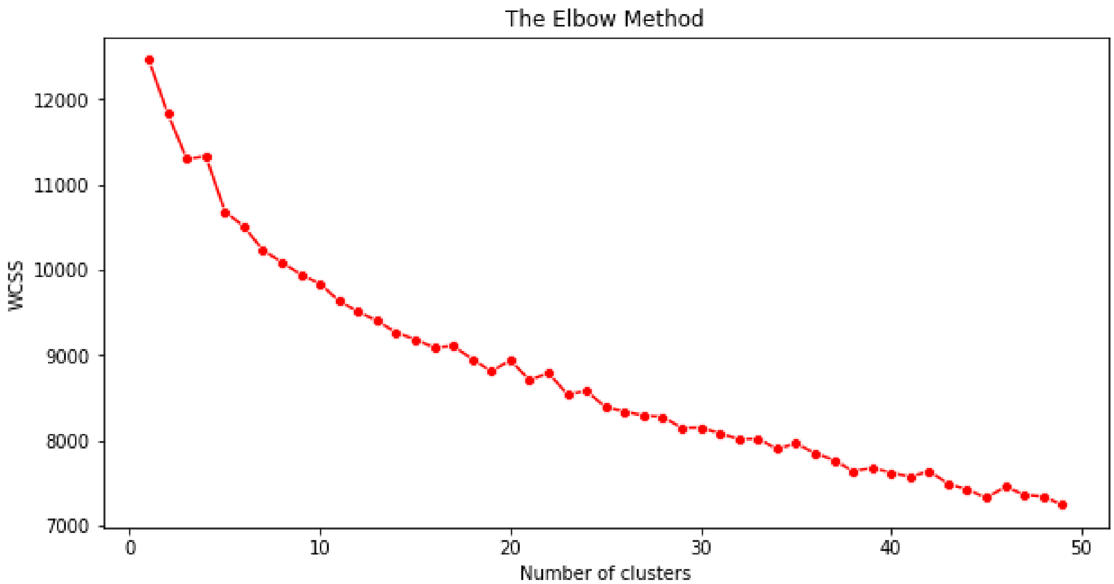
This is exactly the same with our lab exercise =D

[18]:
```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 50):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
```

```
    kmeans.fit(X)
    # inertia method returns wcss for that model
    wcss.append(kmeans.inertia_)
```

Plot the Elbow Diagram

[19]:
```
plt.figure(figsize=(10,5))
sns.lineplot(range(1, 50), wcss,marker='o',color='red')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



From the Elbow Method above, k = 5 is choosen

[20]:
```
n_clusters = 5
matrix = X.as_matrix()
kmeans = KMeans(n_clusters=n_clusters, init='k-means++', n_init=1)
kmeans.fit(matrix)
#y_kmeans = kmeans.fit_predict(matrix)
clusters = kmeans.predict(matrix)
pd.Series(clusters).value_counts()
```

```
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:2: FutureWarning: Method .as_matrix will be
removed in a future version. Use .values instead.
```

[20]: 4   3402
3   348
2   333
0   184
1   163
dtype: int64

Check the number elements in each Cluster

[21]:
```
pd.Series(clusters).value_counts()
```

[21]:
```
4    3402
3     348
2     333
0     184
1     163
dtype: int64
```

3.4 Silhouette intra-cluster score

In order to have an insight on the quality of the classification, we can represent the silhouette scores of each element of the different clusters. This is the purpose of the next figure which is taken from the sklearn documentation:

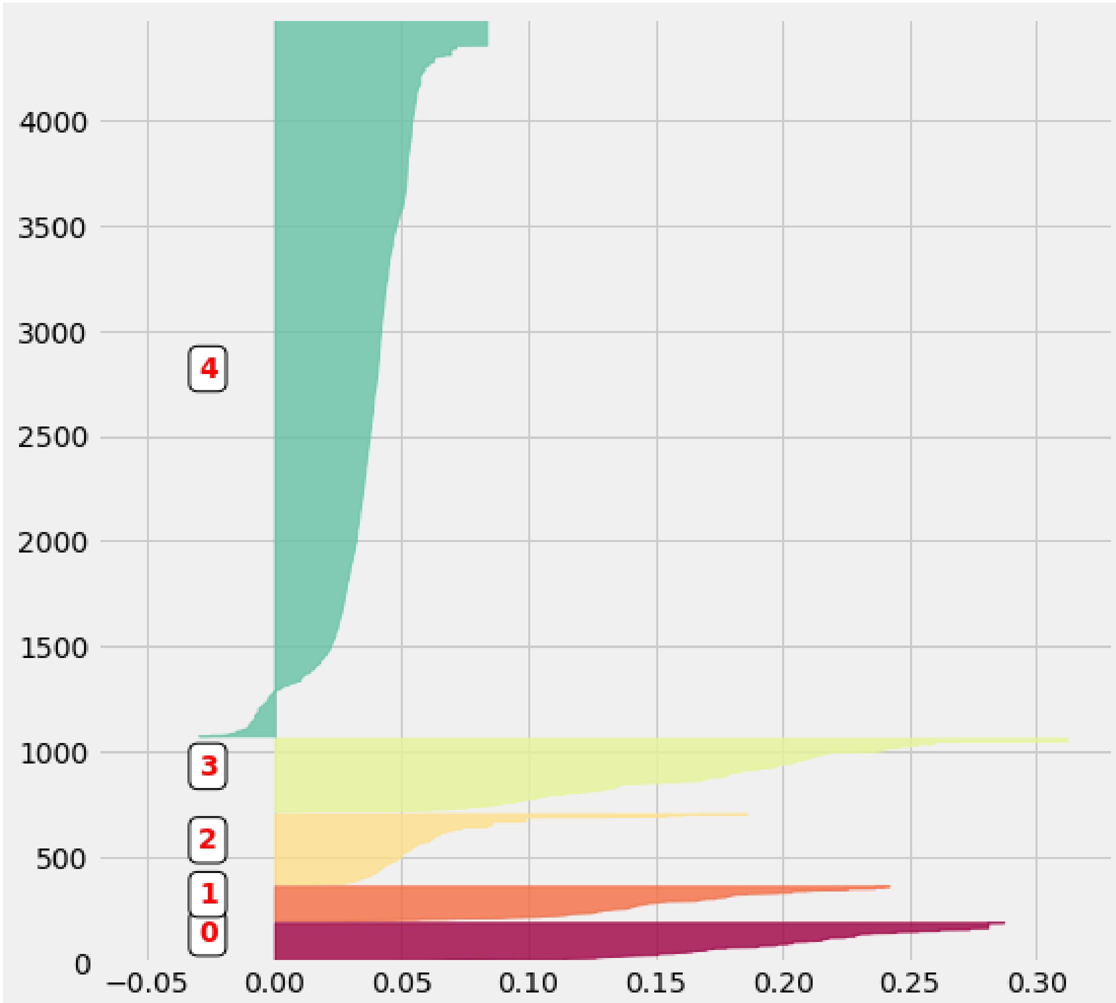
[22]:
```
def graph_component_silhouette(n_clusters, lim_x, mat_size,␣
↪sample_silhouette_values, clusters):
    plt.rcParams["patch.force_edgecolor"] = True
    plt.style.use('fivethirtyeight')
    mpl.rc('patch', edgecolor = 'dimgray', linewidth=1)
    #____________________________
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(8, 8)
    ax1.set_xlim([lim_x[0], lim_x[1]])
    ax1.set_ylim([0, mat_size + (n_clusters + 1) * 10])
    y_lower = 10
    for i in range(n_clusters):
        ␣
↪#___________________________________________________________________________________
        # Aggregate the silhouette scores for samples belonging to cluster i,␣
↪and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[clusters == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        cmap = cm.get_cmap("Spectral")
        color = cmap(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0,␣
↪ith_cluster_silhouette_values,
```

```
                              facecolor=color, edgecolor=color, alpha=0.8)
        #______________________________________________________________________
        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.03, y_lower + 0.5 * size_cluster_i, str(i), color = 'red',␣
 ↪fontweight = 'bold',
                bbox=dict(facecolor='white', edgecolor='black',␣
 ↪boxstyle='round, pad=0.3'))
        #______________________________________
        # Compute the new y_lower for next plot
        y_lower = y_upper + 10
```

[23]:

```
#____________________________________
# define individual silouhette scores
sample_silhouette_values = silhouette_samples(matrix, clusters)
print(sample_silhouette_values)
print(clusters)
#___________________
# and do the graph
graph_component_silhouette(n_clusters, [-0.07, 0.33], len(X),␣
 ↪sample_silhouette_values, clusters)
```

```
[0.17110697 0.04104678 0.04104678 … 0.0404704  0.04005284 0.03420523]
[3 4 4 … 4 4 4]
```



4 Customer Segmentation

4.1 RFM Analysis

The customer transaction dataset held by the U.K. merchant has 5 variables as shown in table below , and it contains all the transactions occurring in years 2010 and 2011. It makes each individual consumer, and therefore it makes some in-depth analyses in the present study.

Variable Names	Data Types	Description
Customer ID	Nominal	Corresponding to each distinct product category
Recency	Numeric	Recency in month
InvoiceDay	Numeric	Time in month since the first purchase in 2011
Frequency	Numeric	Frequency of purchase per product category
Monetary	Numeric	Total amount spent per product category

I'll use only the subset of the full dataset, taking 30% of samples.

[25]:
```
# use a subset of full data
np.random.seed(306)
df_frac = df_clean.sample(frac = .3).reset_index(drop = True)
```

[26]:
```
# extract year, month and day

# change the string into datetime format
df_frac['InvoiceDate'] = pd.to_datetime(df_frac['InvoiceDate'])

# change the output format of the datetime data
df_frac['InvoiceDay'] = df_frac['InvoiceDate'].dt.strftime('%Y-%m-%d')
df_frac.head()
```

[26]:
```
  Invoice StockCode                         Description  Quantity  \
0  497205    85231B        CINAMMON SET OF 9 T-LIGHTS         1
1  500715    47570B               TEA TIME TABLE CLOTH         2
2  507417    47599A                    PINK PARTY BAGS         6
3  515704     22441  GROW YOUR OWN BASIL IN ENAMEL MUG         8
4  494467     85065             CREAM SWEETHEART TRAYS         1

          InvoiceDate  Price  Customer ID         Country  InvoiceDay
0 2010-02-07 11:57:00   0.85      17841.0  United Kingdom  2010-02-07
1 2010-03-09 14:32:00  10.65      16854.0  United Kingdom  2010-03-09
2 2010-05-09 14:35:00   2.10      13425.0  United Kingdom  2010-05-09
3 2010-07-14 12:31:00   2.10      14640.0  United Kingdom  2010-07-14
4 2010-01-14 14:29:00  12.75      15046.0  United Kingdom  2010-01-14
```

RFM (Recency, Frequency, Monetary) is a very Simple Technique that we can apply it very easy and get the super useful analysis for our Customer Segmentation. Recency is days since the customers made the last purchase and by definition, the lower it is the better. Frequency is the number of transaction in the last 12 months. Monetary value is the total amout of money the customers spent in the last 12 months.

[27]:
```
print('Min : {}, Max : {}'.format(min(df_frac.InvoiceDay), max(df_frac.
 ↪InvoiceDay)))
```

```
Min : 2009-12-01, Max : 2010-12-09
```

The last day of purchase in total is 09 DEC, 2011. To calculate the day periods, let's set one day after the last one, or 10 DEC as a pin date. We will count the diff days with pin_date.

[28]:
```
pin_date = str(max(df_frac.InvoiceDay)) + "  " + str(dt.timedelta(1))
print(pin_date)
```

```
2010-12-09  1 day, 0:00:00
```

[29]:
```
# filter the necessary columns for RFM analysis
uk_data=df_frac[['Customer ID','InvoiceDate','Invoice','Quantity','Price']]
uk_data['TotalPrice'] = uk_data['Quantity'] * uk_data['Price']
uk_data['InvoiceDate'].min(),uk_data['InvoiceDate'].max()
PRESENT = dt.datetime(2011,12,10)
uk_data['InvoiceDay'] = pd.to_datetime(uk_data['InvoiceDate'].dt.
 ↪strftime('%Y-%m-%d'))


uk_data.head()
```

```
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[29]:
```
   Customer ID         InvoiceDate Invoice  Quantity  Price  TotalPrice  \
0      17841.0 2010-02-07 11:57:00  497205         1   0.85        0.85
1      16854.0 2010-03-09 14:32:00  500715         2  10.65       21.30
2      13425.0 2010-05-09 14:35:00  507417         6   2.10       12.60
3      14640.0 2010-07-14 12:31:00  515704         8   2.10       16.80
4      15046.0 2010-01-14 14:29:00  494467         1  12.75       12.75

  InvoiceDay
0 2010-02-07
1 2010-03-09
2 2010-05-09
3 2010-07-14
4 2010-01-14
```

[30]:
```
# Create total spend dataframe
uk_data['TotalSum'] = uk_data.Quantity * uk_data.Price
```

```
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[31]:
```
# calculate RFM values
rfm= uk_data.groupby('Customer ID').agg({'InvoiceDate': lambda date: (PRESENT -␣
 ↪date.max()).days,
                                         'Invoice': lambda num: len(num),
                                         'TotalPrice': lambda price: price.
 ↪sum()})
# rfm= uk_data.agg({'InvoiceDate': lambda date: (PRESENT - date.max()).days,
#                                          'Invoice': lambda num: len(num),
#                                          'TotalPrice': lambda price: price.
 ↪sum()})
rfm
```

[31]:
```
             InvoiceDate  Invoice  TotalPrice
Customer ID
12346.0              527       19      158.78
12608.0              404        3       97.20
12745.0              486       11      489.22
12746.0              527        4       37.15
12747.0              369       46     1324.93
…                      …        …           …
18283.0              382       61      180.42
18284.0              431       11      155.94
18285.0              660        5      129.90
18286.0              476       26      397.90
18287.0              382       24      619.85

[3863 rows x 3 columns]
```

[32]:
```
rfm.columns
```

[32]:
```
Index(['InvoiceDate', 'Invoice', 'TotalPrice'], dtype='object')
```

As the three columns are grouped by customers and count the days from the max date value, Recency is the days since the last purchase of a customer. Frequency is the number of purchases of a customer and Monetary is the total amount of spend of a customer.

[33]:
```
# Change the name of columns
rfm.columns=['monetary','frequency','recency']
rfm['recency'] = rfm['recency'].astype(int)
rfm.head()
```

[33]:

Customer ID	monetary	frequency	recency
12346.0	527	19	158
12608.0	404	3	97
12745.0	486	11	489
12746.0	527	4	37
12747.0	369	46	1324

4.2 RFM Quartiles

Let's group the customers based on Recency and Frequency. We will use quantile values to get three equal percentile groups an then make three separate gruops. As the lower Recency value is the better, we will label them in decreasing order.

[34]:
```
# create labels and assign them to tree percentile groups
r_labels = range(4, 0, -1)
r_groups = pd.qcut(rfm.recency, q = 4, labels = r_labels)

f_labels = range(1, 5)
f_groups = pd.qcut(rfm.frequency, q = 4, labels = f_labels)

m_labels = range(1, 5)
m_groups = pd.qcut(rfm.monetary, q = 4, labels = m_labels)

m_groups.head()
```

[34]:
```
Customer ID
12346.0    4
12608.0    2
12745.0    3
12746.0    4
12747.0    1
Name: monetary, dtype: category
Categories (4, int64): [1 < 2 < 3 < 4]
```

[35]:
```
# make a new column for group labels
rfm['R'] = r_groups.values
rfm['F'] = f_groups.values
rfm['M'] = m_groups.values
```

[36]:
```
# sum up the three columns
rfm['RFM_Segment'] = rfm.apply(lambda x: str(x['R']) + str(x['F']) +␣
 ↪str(x['M']), axis = 1)
rfm['RFM_Score'] = rfm[['R', 'F', 'M']].sum(axis = 1)
rfm.head()
```

[36]:

	monetary	frequency	recency	R	F	M	RFM_Segment	RFM_Score
Customer ID								
12346.0	527	19	158	3	3	4	334	10.0
12608.0	404	3	97	3	1	2	312	6.0
12745.0	486	11	489	2	2	3	223	7.0
12746.0	527	4	37	4	1	4	414	9.0
12747.0	369	46	1324	1	4	1	141	6.0

With this value, we can go further analysis such as what is the average values for each RFM values or leveling customers in total RFM score. RFM scores was chosen as input for the clustering analysis. RFM Segment finally assists to interpret each RFM scores found with top best ten customers. This segmentation by three clusters seems to have a clearer interpretation of the target dataset than the ones by four levels.

RFM Score (11.0)- 218.8 Pound

RFM Score (12.0)- 248.6 Pound

RFM Score (12.0)- 369.0 Pound

[37]:

```python
# calculate averae values for each RFM_score
rfm_agg = rfm.groupby('RFM_Score').agg({
    'recency' : 'mean',
    'frequency' : 'mean',
    'monetary' : ['mean', 'count']
})

rfm_agg.round(1).head()
```

[37]:

	recency	frequency	monetary	
	mean	mean	mean	count
RFM_Score				
3.0	777.2	3.5	368.8	4
4.0	713.5	5.9	382.7	33
5.0	710.5	14.6	382.9	195
6.0	1380.9	59.1	384.0	820
7.0	487.0	30.6	410.2	960

The final score will be the aggregated value of RFM and we can make groups based on the RFM_Score

[38]:

```python
# assign labels from total score
score_labels = ['Green', 'Bronze', 'Silver', 'Gold']
score_groups = pd.qcut(rfm.RFM_Score, q = 4, labels = score_labels)
rfm['RFM_Level'] = score_groups.values
#Filter out Top/Best cusotmers
rfm[rfm['RFM_Level']=='Gold'].sort_values('monetary', ascending=False).head(10)
```

[38]:

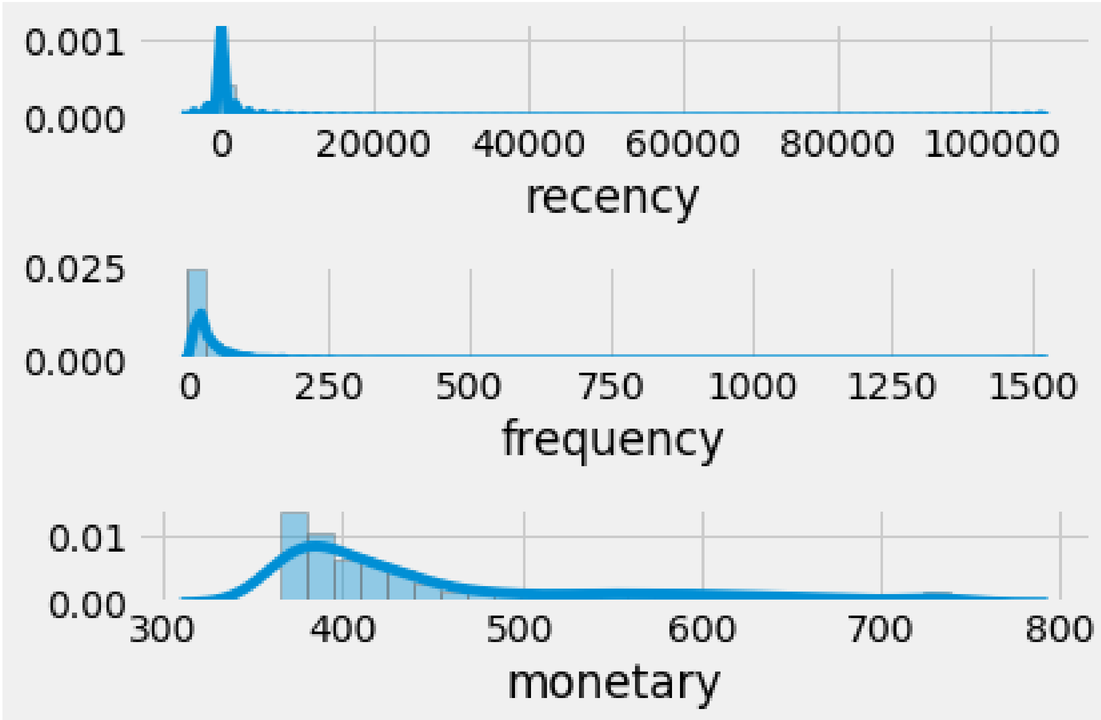
```
             monetary  frequency  recency  R  F  M RFM_Segment  RFM_Score  \
Customer ID
13526.0           738         16      165  3  3  4         334       10.0
17641.0           738          1       -6  4  1  4         414        9.0
17592.0           738          6       27  4  1  4         414        9.0
17485.0           738          3      -12  4  1  4         414        9.0
17056.0           738          1        3  4  1  4         414        9.0
16763.0           738          8      139  3  2  4         324        9.0
14654.0           738         11       69  4  2  4         424       10.0
14980.0           737          7       39  4  2  4         424       10.0
17660.0           737         17       70  4  3  4         434       11.0
13457.0           737          2       26  4  1  4         414        9.0

            RFM_Level
Customer ID
13526.0          Gold
17641.0          Gold
17592.0          Gold
17485.0          Gold
17056.0          Gold
16763.0          Gold
14654.0          Gold
14980.0          Gold
17660.0          Gold
13457.0          Gold
```

4.3 Customer Segmentation with K-Means

We can also apply Kmeans clustering with RFM values. As Kmeans clustering require data to be normalized and has a symmetric distribution, preprocessing process in scale is needed.Recency Frequency and Monetary are value ranges [0, 12], [1, 169]and [3, 125], quite different respectively. As such, these variables should be normalized before the clustering analysis.

[39]:

```python
# plot the distribution of RFM values
plt.subplot(3, 1, 1); sns.distplot(rfm.recency, label = 'recency')
plt.subplot(3, 1, 2); sns.distplot(rfm.frequency, label = 'frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm.monetary, label = 'monetary')

plt.tight_layout()
plt.show()
```



As you can see above, the values are skewed and need to be normalized. Due to the zero or negative values in Recency and MonetaryValue, we need to set them 1 before log transformation and scaling.

[40]:
```
# define function for the values below 0
def neg_to_zero(x):
    if x <= 0:
        return 1
    else:
        return x
```

[41]:
```
# apply the function to Recency and MonetaryValue column
rfm['recency'] = [neg_to_zero(x) for x in rfm.recency]
rfm['monetary'] = [neg_to_zero(x) for x in rfm.monetary]
rfm.head()
```

[41]:
```
             monetary  frequency  recency  R  F  M RFM_Segment  RFM_Score  \
Customer ID
12346.0           527         19      158  3  3  4         334       10.0
12608.0           404          3       97  3  1  2         312        6.0
12745.0           486         11      489  2  2  3         223        7.0
12746.0           527          4       37  4  1  4         414        9.0
12747.0           369         46     1324  1  4  1         141        6.0

            RFM_Level
```

```
Customer ID
12346.0          Gold
12608.0         Green
12745.0        Bronze
12746.0          Gold
12747.0         Green
```

[42]:

```
# unskew the data
rfm_log = rfm[['recency', 'frequency', 'monetary']].apply(np.log, axis = 1).
 ↪round(3)
rfm_log.describe()
```

[42]:

	recency	frequency	monetary
count	3863.000000	3863.000000	3863.000000
mean	5.275859	2.609010	6.103159
std	1.461534	1.261692	0.195864
min	0.000000	0.000000	5.900000
25%	4.443000	1.792000	5.948000
50%	5.298000	2.639000	6.033000
75%	6.227000	3.497000	6.224000
max	11.582000	7.322000	6.604000

[43]:

```
# scale the data
scaler = StandardScaler()
rfm_scaled = scaler.fit_transform(rfm_log)
```

[44]:

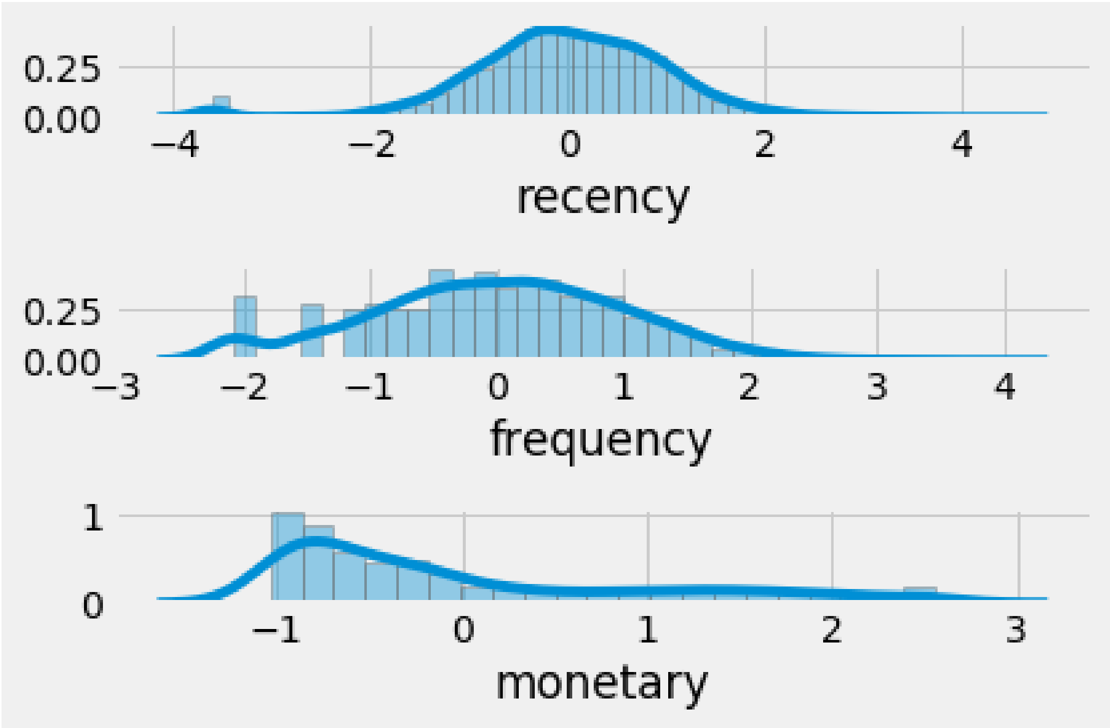
```
# transform into a dataframe
rfm_scaled = pd.DataFrame(rfm_scaled, index = rfm.index, columns = rfm_log.
 ↪columns)
rfm_scaled.head()
```

[44]:

	recency	frequency	monetary
Customer ID			
12346.0	-0.145660	0.265543	0.836610
12608.0	-0.479599	-1.196968	-0.521651
12745.0	0.626916	-0.167265	0.423004
12746.0	-1.139265	-0.969466	0.836610
12747.0	1.308480	0.967072	-0.981213

[45]:

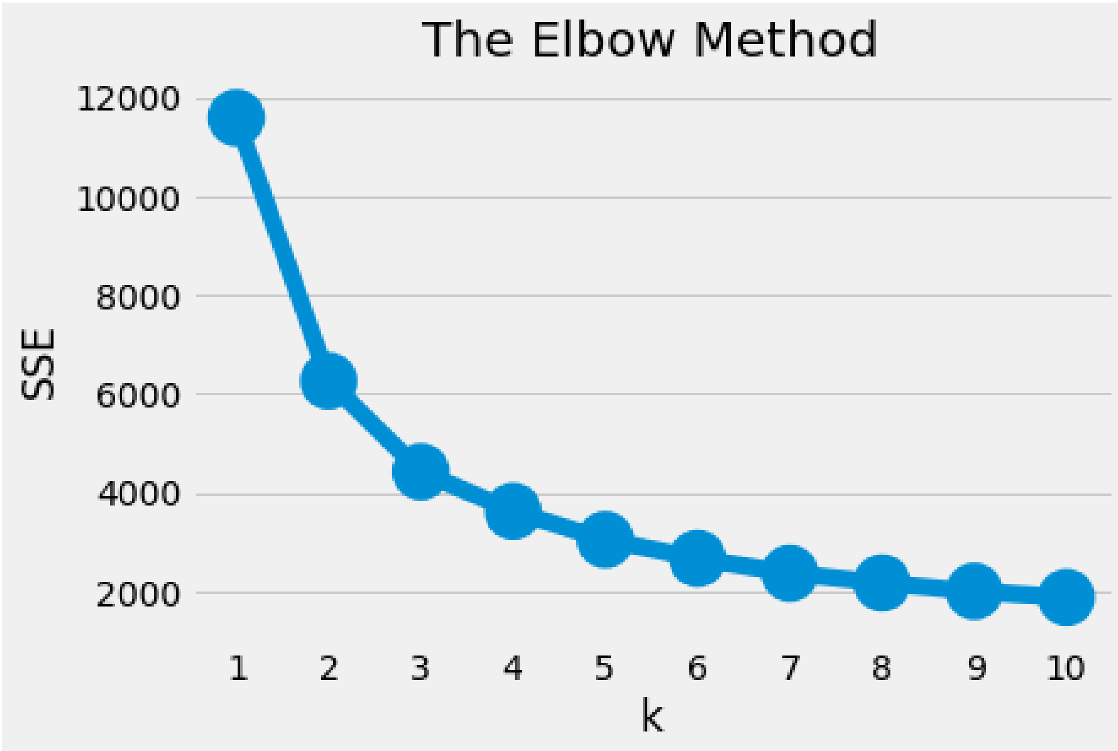
```
# plot the distribution of RFM values
plt.subplot(3, 1, 1); sns.distplot(rfm_scaled.recency, label = 'Recency')
plt.subplot(3, 1, 2); sns.distplot(rfm_scaled.frequency, label = 'Frequency')
plt.subplot(3, 1, 3); sns.distplot(rfm_scaled.monetary, label = 'Monetary')

plt.tight_layout()
plt.show()
```



With the Elbow method, we can get the optimal number of clusters. From the elbow curve below, it is observed that the more popular k are the ones with three units. This attribute is closely correlated to RFM level as the popularity of certain Gold, Silver and Bronze will be reflected in the clusters.

[46]:
```
# initiate an empty dictionary
wcss = {}

# Elbow method with for loop
for i in range(1, 11):
    kmeans = KMeans(n_clusters= i, init= 'k-means++', max_iter= 300)
    kmeans.fit(rfm_scaled)
    wcss[i] = kmeans.inertia_
```

[47]:
```
from matplotlib import pyplot as plt
# Plot SSE for each *k*
plt.title('The Elbow Method')
sns.pointplot(x=list(wcss.keys()), y=list(wcss.values()))
plt.xlabel('k'); plt.ylabel('SSE')
plt.show()
```



[48]:
```
# choose n_clusters = 3
clus = KMeans(n_clusters= 3, init= 'k-means++', max_iter= 300)
clus.fit(rfm_scaled)
```

[48]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

[49]:
```
# Assign the clusters to dataframe
rfm['K_Cluster'] = clus.labels_
rfm.head()
```

[49]:
```
             monetary  frequency  recency  R  F  M RFM_Segment  RFM_Score  \
Customer ID
12346.0           527         19      158  3  3  4         334       10.0
12608.0           404          3       97  3  1  2         312        6.0
12745.0           486         11      489  2  2  3         223        7.0
12746.0           527          4       37  4  1  4         414        9.0
12747.0           369         46     1324  1  4  1         141        6.0

            RFM_Level  K_Cluster
Customer ID
12346.0          Gold          2
```

```
12608.0         Green           2
12745.0        Bronze           2
12746.0          Gold           1
12747.0         Green           0
```

[50]:
```
rfm_final = rfm
rfm_final = rfm[['recency', 'frequency', 'monetary', 'K_Cluster']]
rfm_final.describe()
rfm_final
```

[50]:
```
             recency  frequency  monetary  K_Cluster
Customer ID
12346.0          158         19       527          2
12608.0           97          3       404          2
12745.0          489         11       486          2
12746.0           37          4       527          1
12747.0         1324         46       369          0
…                  …          …         …          …
18283.0          180         61       382          0
18284.0          155         11       431          2
18285.0          129          5       660          1
18286.0          397         26       476          0
18287.0          619         24       382          0

[3863 rows x 4 columns]
```

4.4 Visualization of Customer Segmentation

4.4.1 Snake Plot

In marketing, snail plot and heatmap are often used plot for visualization. I'll use the rfm_scaled dataframe with normalized rfm values for the plot. Overall, Gold and as a result, spent on product with word occurrence

"pink",

"heart" and

"candle"

respectively can be categorized as low recency, high frequency and high monetary with a purchased quite often spending per consumer. Silver can be categorized as medium recency, low frequency and medium monetary with a medium spending per consumer. Bronze can be categorized as high recency, high frequency and low monetary with a regular spending per consumer.

[51]:
```
# assign cluster column
rfm_scaled['K_Cluster'] = clus.labels_
rfm_scaled['RFM_Level'] = rfm.RFM_Level
rfm_scaled.reset_index(inplace = True)
rfm_scaled.head()
```

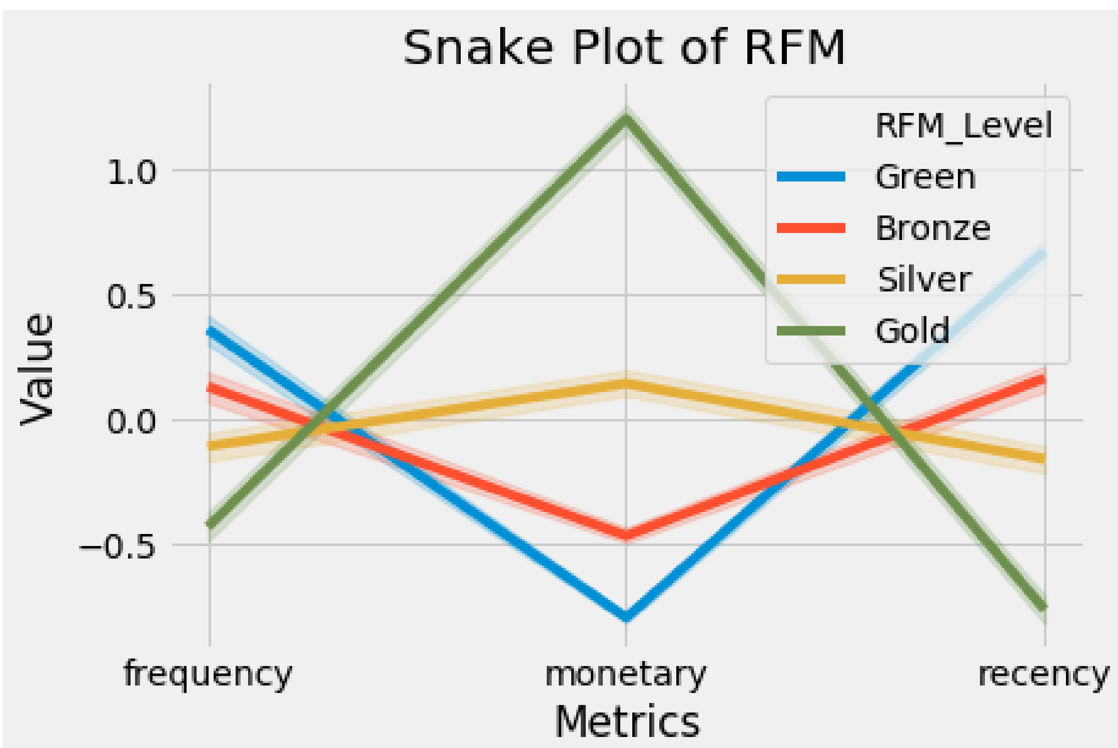
[51]:
```
   Customer ID   recency  frequency  monetary  K_Cluster RFM_Level
0      12346.0 -0.145660   0.265543  0.836610          2      Gold
1      12608.0 -0.479599  -1.196968 -0.521651          2     Green
2      12745.0  0.626916  -0.167265  0.423004          2    Bronze
3      12746.0 -1.139265  -0.969466  0.836610          1      Gold
4      12747.0  1.308480   0.967072 -0.981213          0     Green
```

[52]:
```
# melt the dataframe
rfm_melted = pd.melt(frame= rfm_scaled, id_vars= ['Customer ID', 'RFM_Level',␣
↪'K_Cluster'],
                     var_name = 'Metrics', value_name = 'Value')
rfm_melted.head()
```

[52]:
```
   Customer ID RFM_Level  K_Cluster  Metrics     Value
0      12346.0      Gold          2  recency -0.145660
1      12608.0     Green          2  recency -0.479599
2      12745.0    Bronze          2  recency  0.626916
3      12746.0      Gold          1  recency -1.139265
4      12747.0     Green          0  recency  1.308480
```

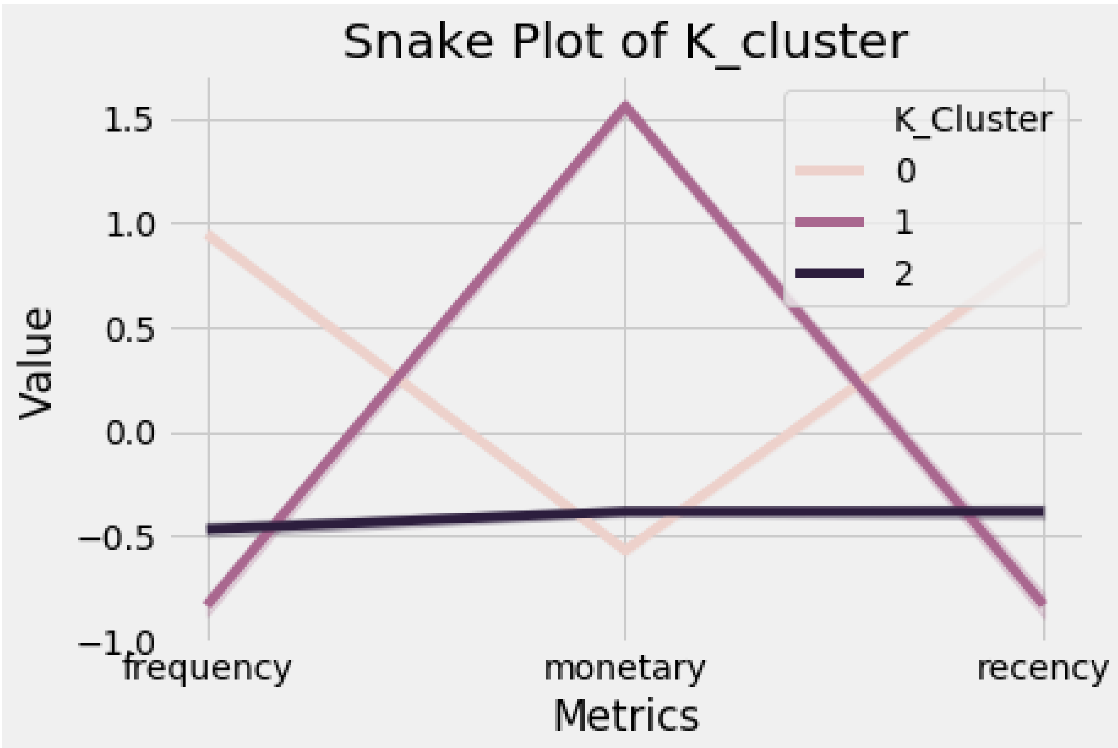
[53]:
```
# a snake plot with RFM
sns.lineplot(x = 'Metrics', y = 'Value', hue = 'RFM_Level', data = rfm_melted)
plt.title('Snake Plot of RFM')
plt.legend(loc = 'upper right')
```

[53]:
```
<matplotlib.legend.Legend at 0x264113f48c8>
```



[54]:
```
# a snake plot with K-Means
sns.lineplot(x = 'Metrics', y = 'Value', hue = 'K_Cluster', data = rfm_melted)
plt.title('Snake Plot of K_cluster')
plt.legend(loc = 'upper right')
```

[54]: <matplotlib.legend.Legend at 0x264139ad588>



4.4.2 HeatMap

Heatmap is efficient for comparing the standardized values.

Cluster 0 relates to some 1900 consumers, composed of 50.0 per cent of the whole population

Cluster 1 relates to some 627 consumers, composed of 17.0 per cent of the whole population

Cluster 2 relates to some 1368 consumers, composed of 36.0 per cent of the whole population

[55]:
```
# Calculate average values of each cluster
cluster_avg = rfm.groupby('RFM_Level').mean().iloc[:, 0:3]
# Calculate average values of population
population_avg = rfm_log.mean()
cluster_avg.head()
```

[55]:

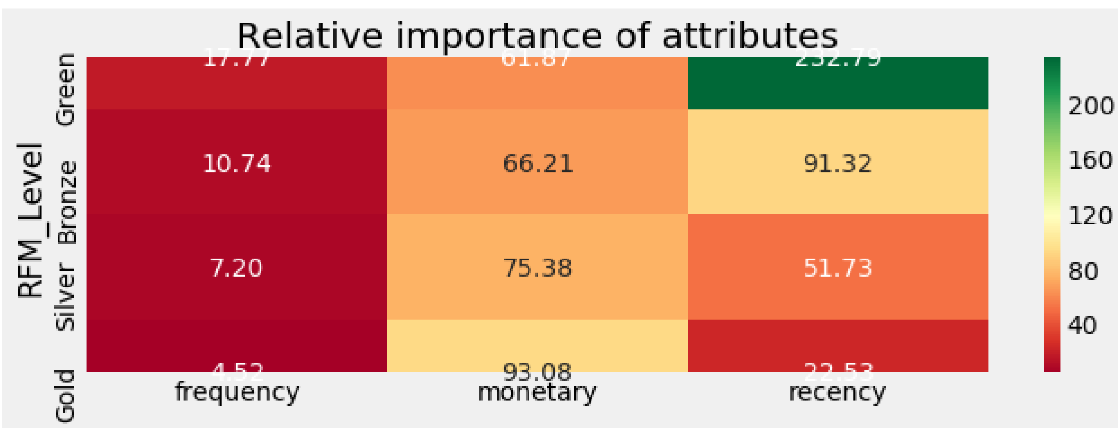
	monetary	frequency	recency
RFM_Level			
Green	383.725285	48.961027	1233.425856
Bronze	410.216667	30.626042	487.085417
Silver	466.147321	21.392857	278.208705
Gold	574.160209	14.398953	124.159162

[56]:
```
# Calculate importance score by dividing them and subtracting 1
relative_imp = cluster_avg / population_avg - 1
relative_imp.round(2)
```

[56]:

	frequency	monetary	recency
RFM_Level			
Green	17.77	61.87	232.79
Bronze	10.74	66.21	91.32
Silver	7.20	75.38	51.73
Gold	4.52	93.08	22.53

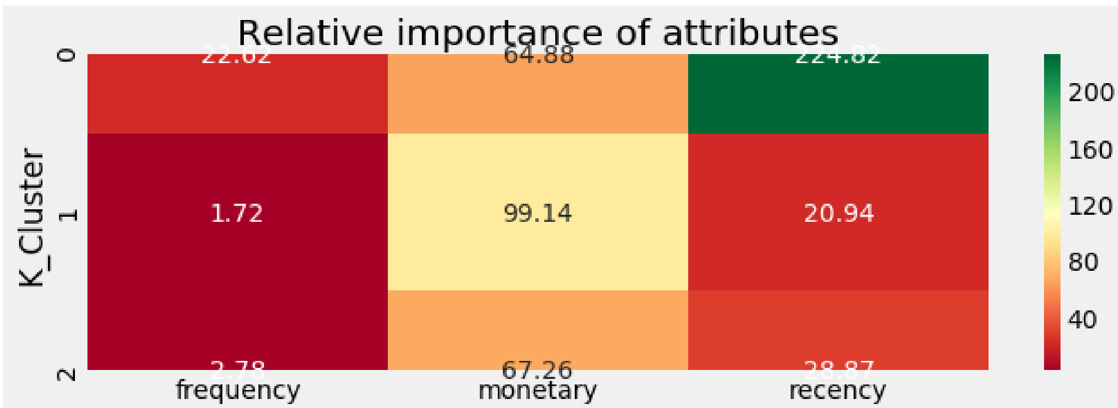
[57]:
```
# Analyze and plot relative importance
plt.figure(figsize=(10, 3))
plt.title('Relative importance of attributes')
sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='RdYlGn')
plt.show()
```



[58]:
```
# Calculate average values of each cluster
cluster_avg_K = rfm.groupby('K_Cluster').mean().iloc[:, 0:3]
# Calculate average values of population
population_avg = rfm_log.mean()
# Calculate importance score by dividing them and subtracting 1
relative_imp = cluster_avg_K / population_avg - 1
relative_imp.round(2)
```

```
[58]:             frequency  monetary  recency
      K_Cluster
      0               22.62     64.88   224.82
      1                1.72     99.14    20.94
      2                2.78     67.26    28.87
```

```python
[59]: # Analyze and plot relative importance
      plt.figure(figsize=(10, 3))
      plt.title('Relative importance of attributes')
      sns.heatmap(data=relative_imp, annot=True, fmt='.2f', cmap='RdYlGn')
      plt.show()
```



5 Data Modelling

The type of classifier or machine learning algorithm we choose to implement in this project can be divided into parametric and non-parametric: 1. Parametric : Support Vector Machine Classifier, Naïve Bayes *2. Non-Parametric: Decision Tree, Naïve Bayes*

5.0.1 Classifier Selection

The purpose here is to select one classifier each from parametric and non-parametric machine learning algorithms.

5.0.2 Parametric Machine Learning Algorithms

Decision: Both algorithms selected. Pne of the reasons is both works well with high dimension dataset. Both the training process is less expensive and less time consuming because of the small dataset. According to research done by Miriam, Pedro and Ardelia, ANN has proven record in predicting mortality in HCC patients. However, there is no reason to reject SVM as well.

5.0.3 Non-Parametric Machine Learning Algorithms

Naïve Bayes classifier is selected because: - Naïve Bayes used all the available information in making the decision. It takes into account of all the attributes which don't waste any useful data. This make it a good classifier for medical data which has relatively small dataset. - Handle well high dimensional dataset. It can deal with well separated categories. A Gaussion Naïve Bayes able to handler continuous data. - It is based on the posterior probability of each features. Therefore, it is easy to understand which features are influencing the predictions.

[61]:
```
# assign cluster column
rfm_int = rfm
```

Remove the GROUPBY from RFM analysis

[62]:
```
rfm_int['K_Cluster'] = clus.labels_
```

[63]:
```
rfm_int.reset_index(inplace = True)
```

New dataframe of 3 features

[64]:
```
rfm_final = rfm_int[['recency', 'frequency', 'monetary', 'K_Cluster']]
rfm_final.head()
print(rfm_final)
```

```
      recency  frequency  monetary  K_Cluster
0         158         19       527          2
1          97          3       404          2
2         489         11       486          2
3          37          4       527          1
4        1324         46       369          0
…         …          …        …          …
3858      180         61       382          0
3859      155         11       431          2
3860      129          5       660          1
3861      397         26       476          0
3862      619         24       382          0

[3863 rows x 4 columns]
```

Split the data into train and test set. We used ratio of 80:20.

[66]:
```
from sklearn.model_selection import train_test_split

X = rfm_final.iloc[:,:-1]
Y = rfm_final.iloc[:,3]

x_train, x_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8)
x_train
```

[66]:
```
      recency  frequency  monetary
766        20          1       423
810      1400         72       366
1606       68          2       611
```

```
3223          5          1        385
870         186         11        410
…           …          …         …
2547       1087         92        390
1850         70          1        444
2619       1938        165        368
1383         73          4        606
1389        142          7        737

[3090 rows x 3 columns]
```

Get the categorical data of K_Cluster. The class will be used in ploting later.

[67]:
```
TypeCluster = pd.Categorical(rfm_final['K_Cluster'].unique())
```

5.1 Decision Tree Classifier

Several parameters is tried out to optimize the classifier

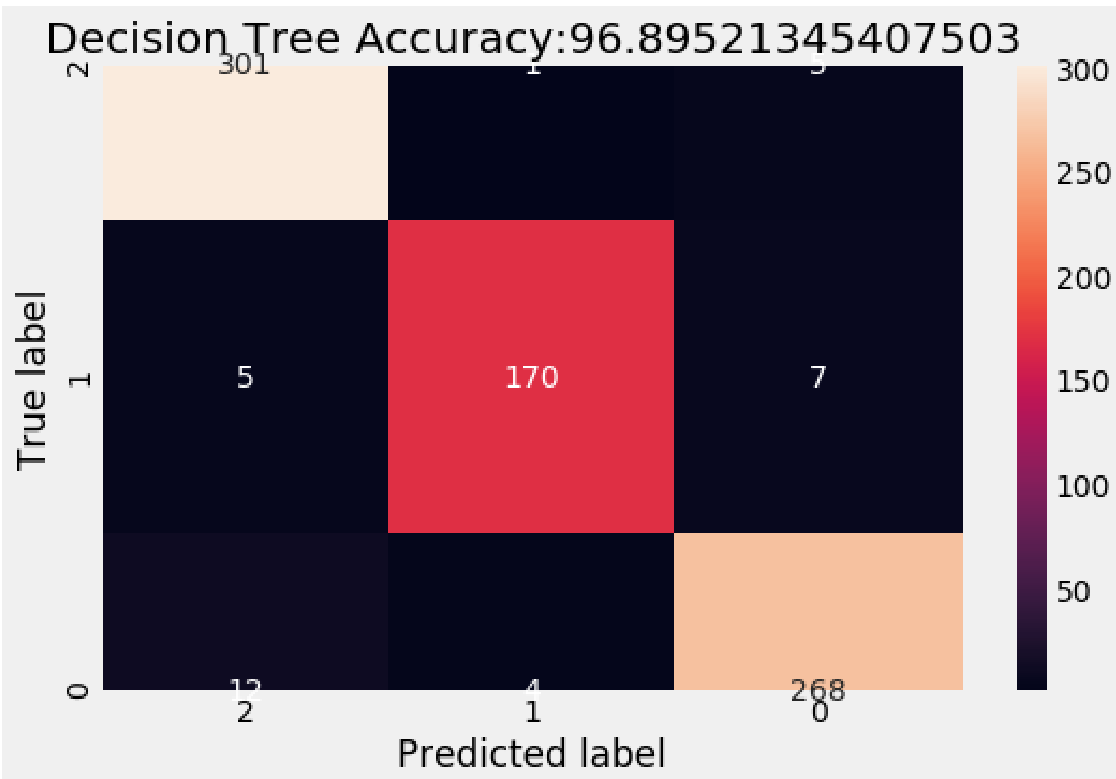
- **criterion**: This parameter allows us to use the different-different attribute selection measure. We choose "entropy" for the information gain.
- **max_depth**: This parameter allows us to choose the split strategy. We choose max_depth=5 because it does not compromise the accuracy of the classifier. ***Note: This is a post-pruning decision, as I already ran the classifier and have the result of max_depth=none. From there, We decided to use max_depth=5.*** The second reason is to reduce the complexity of the tree.
- **splitter**: The maximum depth of the tree. We left it as default, which is "best" to allows the best split at each node.

[68]:
```
# Input the decision tree classifier with parameters
dtree = DecisionTreeClassifier().fit(x_train, y_train)
dtreePruned = DecisionTreeClassifier(criterion='entropy', max_depth=5).
↪fit(x_train, y_train)

print("The prediction accuracy of full dtree is  : {0:2.2f}{1:s}".
↪format(dtreePruned.score(x_test,y_test)*100,"%"))
print("The prediction accuracy of pruned dtree is: {0:2.2f}{1:s}".format(dtree.
↪score(x_test,y_test)*100,"%"))

#####################################################
# Decision Tree with pruning give consistent result
#####################################################

# predict the classes of new, unseen data
y_predictDtree = dtreePruned.predict(x_test)

# Creates a confusion matrix
cmDtree = confusion_matrix(y_test, y_predictDtree)
```

```python
# plot the confusion matrix
plt.figure(figsize=(8,5))
sns.heatmap(cmDtree.T, annot=True, fmt='d',
            xticklabels=TypeCluster,
            yticklabels=TypeCluster)
plt.title("Decision Tree Accuracy:" + str(dtree.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
The prediction accuracy of full dtree is  : 95.60%
The prediction accuracy of pruned dtree is: 96.90%
```

[68]:
```
Text(0.5, 3.6999999999999815, 'Predicted label')
```



Decision Tree classifier with pruning applied yield almost similar performance with the unprune one. On top of that, the whole tree is much more slimmer after pruning.

[69]:
```python
print(classification_report(y_test, y_predictDtree))
```

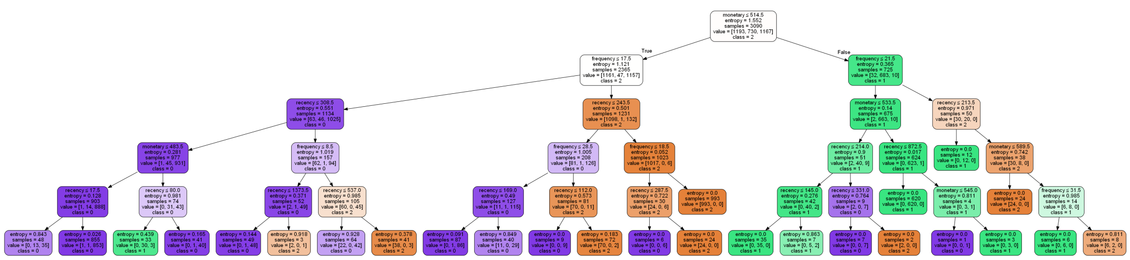
```
              precision    recall  f1-score   support

           0       0.98      0.95      0.96       318
```

```
           1       0.93      0.97      0.95       175
           2       0.94      0.96      0.95       280

    accuracy                           0.96       773
   macro avg       0.95      0.96      0.96       773
weighted avg       0.96      0.96      0.96       773
```

The unprune tree has up to 13 layer of branches before coming to the leaf. After pruning, it become 5 layers.

[70]:
```python
dot_data = StringIO()

# Visualize the whole decision tree.
columns = list(x_train.columns)
#export_graphviz(dtree,
 ↪out_file=dot_data,feature_names=columns,class_names=['2','1','0'],
 ↪filled=True, rounded=True, special_characters=True)
export_graphviz(dtreePruned,
 ↪out_file=dot_data,feature_names=columns,class_names=['2','1','0'],
 ↪filled=True, rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

[70]:



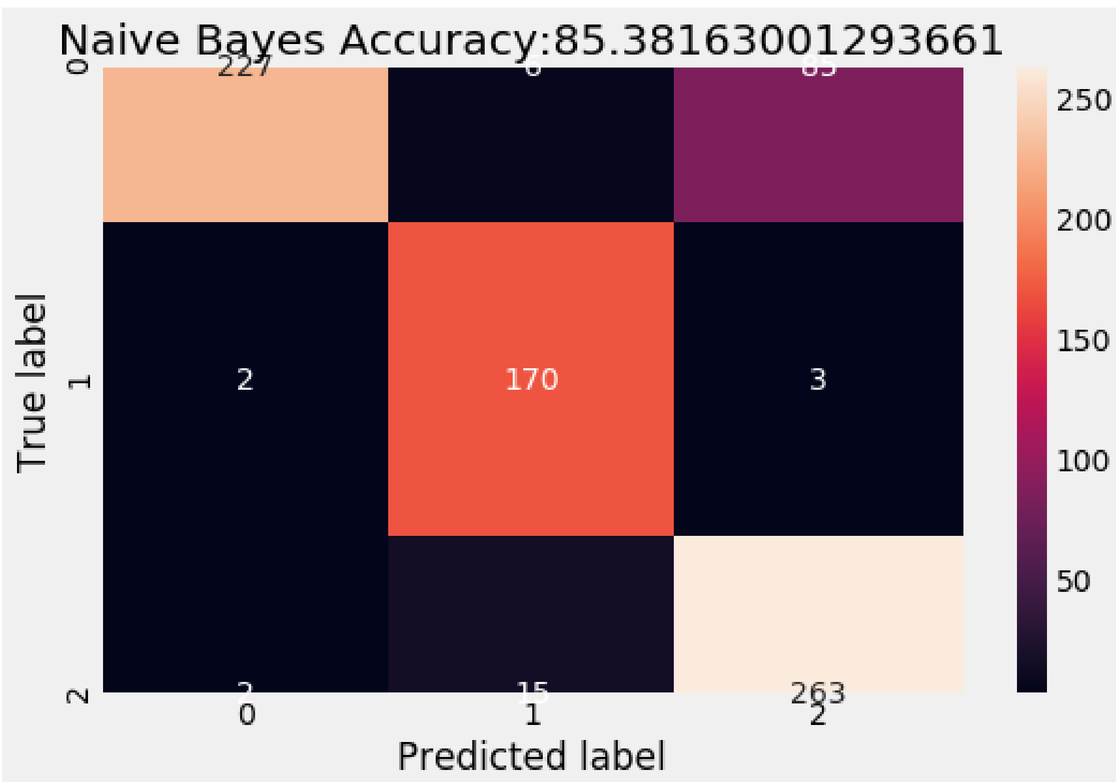
5.2 Naïve Bayes Classifier

Gaussion Naive Bayes is selected.

[71]:
```python
# Instantiate the classifier
gnb = GaussianNB()
# Train classifier
gnb.fit(x_train,y_train)

# Test the classifier
y_predictGnb = gnb.predict(x_test)

# Print results
```

```
print("Number of mislabeled points out of a total {} points : {}, performance {:
→05.2f}%".format(x_test.shape[0], (y_test != y_predictGnb).sum(), gnb.
→score(x_test,y_test)*100 ))

# Creates a confusion matrix
cmGnb = confusion_matrix(y_test, y_predictGnb)

# plot the confusion matrix
plt.figure(figsize=(8,5))
sns.heatmap(cmGnb, annot=True, fmt='g')
plt.title("Naive Bayes Accuracy:" + str(gnb.score(x_test,y_test)*100))
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
Number of mislabeled points out of a total 773 points : 113, performance 85.38%
```



Print the classification report.

```
[72]: print(classification_report(y_test, y_predictGnb))
```

```
              precision    recall  f1-score   support
```

```
              0       0.98      0.71      0.83       318
              1       0.89      0.97      0.93       175
              2       0.75      0.94      0.83       280

       accuracy                           0.85       773
      macro avg       0.87      0.87      0.86       773
   weighted avg       0.88      0.85      0.85       773
```

5.3 Support Vector Machine

Constructed three SVM kernels that are commonly used in machine learning: ``rbf'', ``linear'' and ``sigmoid''.

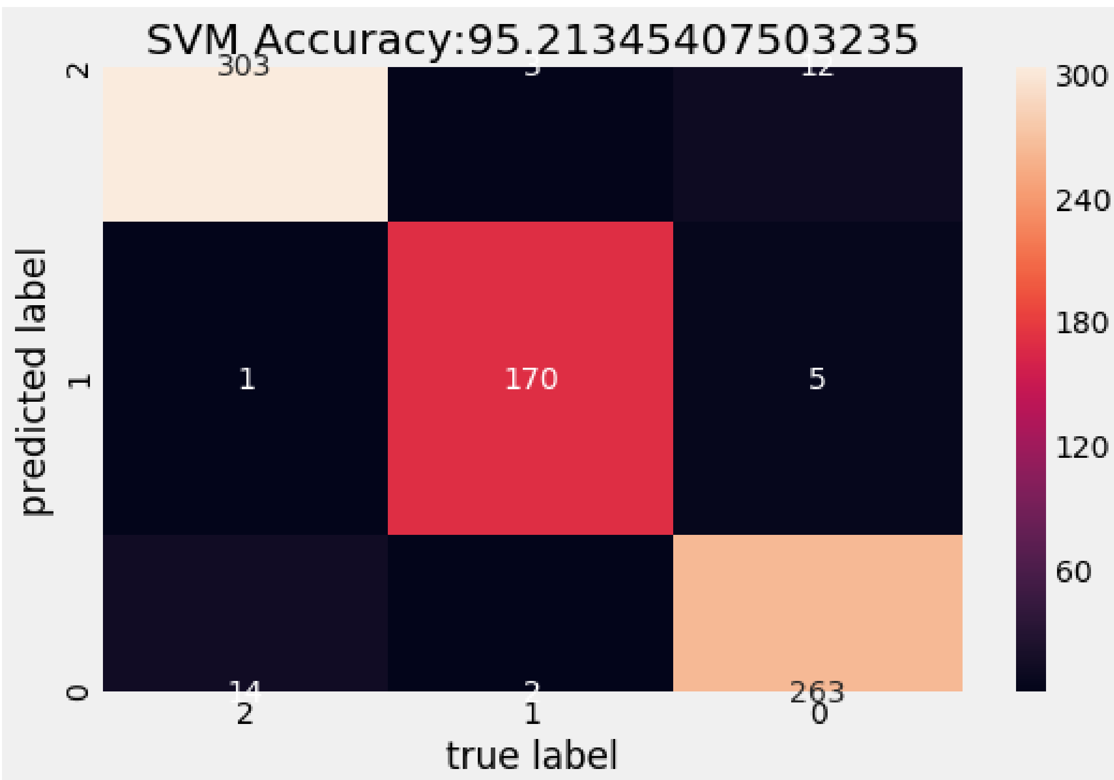
[73]:

```
SvmLinear = svm.SVC(kernel='linear')
SvmRbf = svm.SVC(kernel='rbf')
# SvmPoly = svm.SVC(kernel='poly', degree=5)
SvmSigmoid = svm.SVC(kernel='sigmoid')

SvmLinear.fit(x_train, y_train)
SvmRbf.fit(x_train, y_train)
# SvmPoly.fit(x_train, y_train)
SvmSigmoid.fit(x_train, y_train)

print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmLinear.
 ↪score(x_test,y_test)*100,"%"))
print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmRbf.
 ↪score(x_test,y_test)*100,"%"))
# print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmPoly.
 ↪score(x_test,y_test)*100,"%"))
print("The prediction accuracy is: {0:2.2f}{1:s}".format(SvmSigmoid.
 ↪score(x_test,y_test)*100,"%"))

#############################################
# SVM with linear gives best result
#############################################
y_predictSvm=SvmLinear.predict(x_test)

cm = confusion_matrix(y_test,y_predictSvm)
# plot the confusion matrix
plt.figure(figsize=(8,5))
sns.heatmap(cm.T, annot=True, fmt='d',
            xticklabels=TypeCluster,
            yticklabels=TypeCluster)
plt.title("SVM Accuracy:" + str(SvmLinear.score(x_test,y_test)*100))
plt.xlabel('true label')
plt.ylabel('predicted label')
```

```
plt.show()
```

```
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
C:\Users\tiliew\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\svm\base.py:193: FutureWarning: The default value of gamma will
change from 'auto' to 'scale' in version 0.22 to account better for unscaled
features. Set gamma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

```
The prediction accuracy is: 95.21%
The prediction accuracy is: 50.84%
The prediction accuracy is: 41.14%
```



Print the classification report.

[74]:
```
print(classification_report(y_test, y_predictSvm))
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95       318
```

```
           1       0.97      0.97      0.97       175
           2       0.94      0.94      0.94       280

    accuracy                           0.95       773
   macro avg       0.95      0.95      0.95       773
weighted avg       0.95      0.95      0.95       773
```

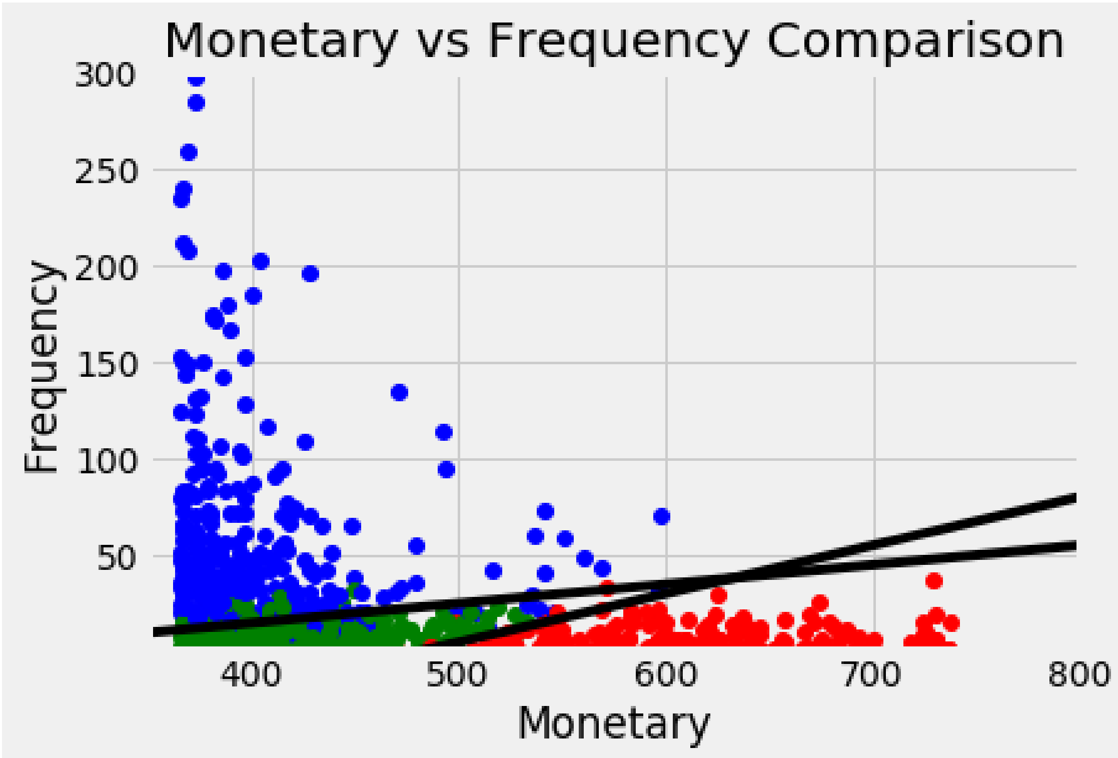
Get the individual attributes for plotting purposes

[75]:
```
SvmLinear2 = svm.SVC(kernel='linear').fit(X, Y)
X_rfm = rfm_final['frequency']
Y_rfm = rfm_final['monetary']
Z_rfm = rfm_final['recency']
```

Plot of Monetary vs Frequency

[76]:
```
colors = {1:'r', 2:'g', 0:'b'}
m1 = 0.1
m2 = 0.25
b1 = -25
b2 = -120
d1=0.15
lineFit = np.linspace(1, 800)
for i in range(len(x_test['frequency'])):
    plt.title('Monetary vs Frequency Comparison')
    plt.xlabel('Monetary')
    plt.ylabel('Frequency')
    plt.scatter( Y_rfm[i], X_rfm[i] , c=colors[rfm_final['K_Cluster'][i]])
plt.plot(lineFit, m1 * lineFit + b1, '-k')
plt.plot(lineFit, m2 * lineFit + b2, '-k')

# plot y = mx + b
yfit = m1 * lineFit + b1
plt.fill_between(lineFit, yfit - d1, yfit + d1, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)
plt.xlim(350, 800)
plt.ylim(1, 300)
```

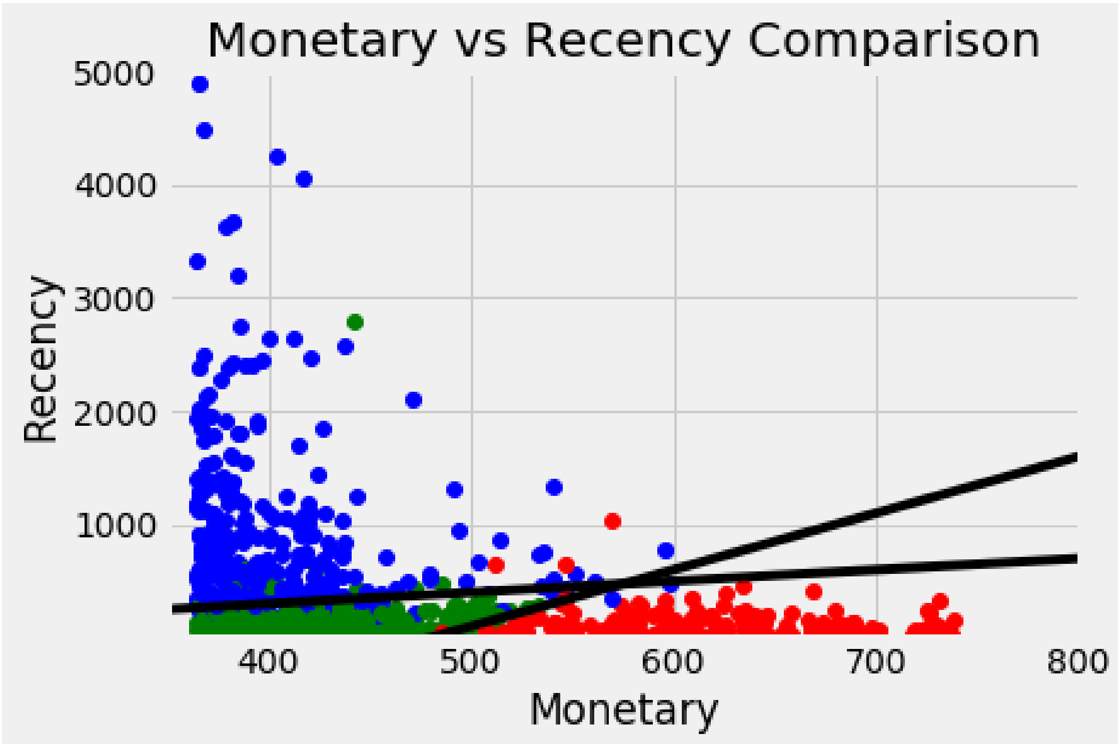
[76]: (1, 300)



Plot of Monetary vs Recency

[77]:
```
m1 = 1
m2 = 5
b1 = -100
b2 = -2400
d1=0.15
lineFit = np.linspace(1, 800)
for i in range(len(x_test['frequency'])):
    plt.title('Monetary vs Recency Comparison')
    plt.xlabel('Monetary')
    plt.ylabel('Recency')
    plt.scatter( Y_rfm[i], Z_rfm[i] , c=colors[rfm_final['K_Cluster'][i]])
plt.plot(lineFit, m1 * lineFit + b1, '-k')
plt.plot(lineFit, m2 * lineFit + b2, '-k')

# plot y = mx + b
yfit = m1 * sepalFit + b1
plt.fill_between(lineFit, yfit - d1, yfit + d1, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)
plt.xlim(350, 800)
plt.ylim(1, 5000)
```

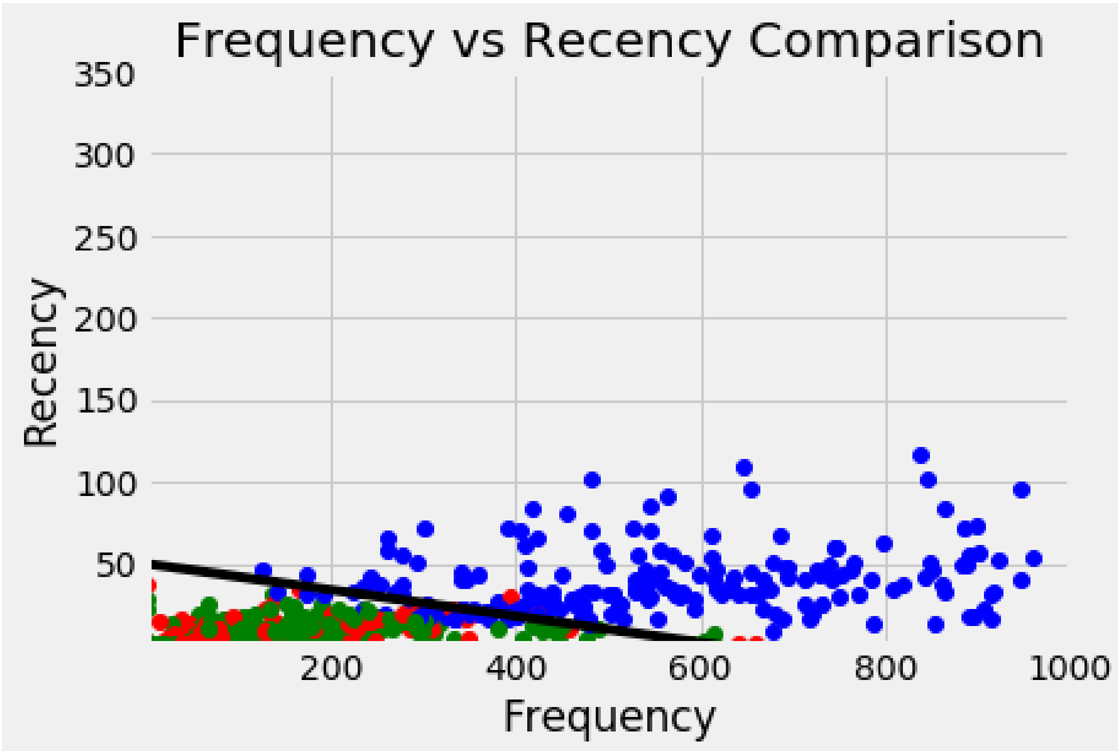
[77]: (1, 5000)



Plot of Frequency vs Recency

[87]:
```
m1 = -0.08
m2 = 0.25
b1 = 50
b2 = -120
d1=0.15
lineFit = np.linspace(1, 800)
for i in range(len(x_test['frequency'])):
    plt.title('Frequency vs Recency Comparison')
    plt.xlabel('Frequency')
    plt.ylabel('Recency')
    plt.scatter( Z_rfm[i] ,X_rfm[i],  c=colors[rfm_final['K_Cluster'][i]])
plt.plot(lineFit, m1 * lineFit + b1, '-k')
#plt.plot(sepalFit, m2 * sepalFit + b2, '-k')

# plot y = mx + b
yfit = m1 * lineFit + b1
plt.fill_between(lineFit, yfit - d1, yfit + d1, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)
plt.xlim(1, 1000)
plt.ylim(1, 350)
```

[87]: (1, 350)



6 Testing and Validation

5-fold cross-validation is conducted on Decision Tree Classifier, Naïve Bayes Classifier and Support Vector Machine to evaluate on their performance.

[89]:
```
# get score for the 10 fold cross validation
scoreDtree = cross_val_score(dtreePruned, x_train, y_train, cv=5,␣
 ↪scoring='accuracy')
scoreGnb = cross_val_score(gnb, x_train, y_train, cv=5, scoring='accuracy')
scoreSvm = cross_val_score(SvmLinear, x_train, y_train, cv=5,␣
 ↪scoring='accuracy')
print("The 5-fold cross-validation score for Decision Tree Classifieris: "  +␣
 ↪str(scoreDtree.mean()*100)+ "%")
print("The 5-fold cross-validation score for Naive Bayes is            : "  +␣
 ↪str(scoreGnb.mean()*100)+ "%")
print("The 5-fold cross-validation score for Support Vector Machine is : "  +␣
 ↪str(scoreSvm.mean()*100)+ "%")
```

```
The 5-fold cross-validation score for Decision Tree Classifieris:
96.31077671860304%
The 5-fold cross-validation score for Naive Bayes is            :
```

87.11861437157307%
The 5-fold cross-validation score for Support Vector Machine is :
96.56972884629053%

The result of **Accuracy** is summarized in table below:

	Decision Tree	Naïve Bayes	SVM
Normal accu.	95.60%	87.12%	95.21%
5-fold CV	96.25%	85.38%	96.57%

The cross-validation result shows that the most performing machine learning algorithm is Decision Tree classifier using the dataset from RFM analysis at 96.25%. The result is followed by Naïve Bayes at 87.12%. Support Vector Machine however on the other scored less than 50% and the cross validation result proved that SVM is not a reliable algorithm to be use in this project.

7 Conclusion

1. The online detail data set is cleaned to keep the UK only data, removed missing, irrelevant and NAs in rows.
2. RFM Analysis is performed to cluster the customer into 3 categories by using K-Means Clustering with aid of Elbow Method.
3. In data modelling, Decision Tree Classifier, Naïve Bayes Classifier and Support Vector Machine is used. The result of each classifier is significant. Decision Tree appears to be the best classifier with 95.60% in accuracy with the support of consistent cross validation result. The performance is followed by Naive Bayes Classifier at 87.12% and consistent cross validation result. The worst performing algorithm is Support Vector Machine with accuracy at 46.83%.

[ ]:

```

```