POLIST_02_notebookessais

September 8, 2021

[1]:
```
import numpy as np
import pandas as pd

from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score,␣
 ↪adjusted_rand_score

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.cm as cm

%matplotlib inline
import seaborn as sns

import datetime
from dateutil.relativedelta import *
import scipy.stats as stats
import math as mt
import timeit

from sklearn import decomposition, preprocessing

import plotly.graph_objects as go
import plotly.express as px
```

Reading input data customers = pd.read_csv("./data/olist_customers_dataset.csv") geoloc = pd.read_csv("./data/olist_geolocation_dataset.csv") items = pd.read_csv("./data/olist_order_items_dataset.csv") payments = pd.read_csv("./data/olist_order_payments_dataset.csv") reviews = pd.read_csv("./data/olist_order_reviews_dataset.csv") orders = pd.read_csv("./data/olist_orders_dataset.csv")

[2]:
```
customers = pd.read_csv("./data/customers.csv")
#geoloc = pd.read_csv("./data/olist_geolocation_dataset.csv")
items = pd.read_csv("./data/items.csv")
payments = pd.read_csv("./data/payments.csv")
reviews = pd.read_csv("./data/reviews.csv")
```

```
orders = pd.read_csv("./data/orders.csv")
```

[3]:
```
#convert ['order_purchase_timestamp','order_approved_at',␣
↪'order_delivered_carrier_date','order_delivered_customer_date',␣
↪'order_estimated_delivery_date'] to datetime
date_columns = ['order_purchase_timestamp','order_approved_at',␣
↪'order_delivered_carrier_date','order_delivered_customer_date',␣
↪'order_estimated_delivery_date']
for col in date_columns:
    orders[col] = pd.to_datetime(orders[col])
```

[24]:
```
# function transforming timedelta values to seconds
def to_seconds(x):
    return x.total_seconds()
```

0.0.1 Constitution du score RFM

Data aggregation

[4]:
```
#Processing related to "orders"

order_custom = orders.merge(customers[['customer_id', 'customer_unique_id']],␣
↪on='customer_id')
part1 = order_custom[['order_purchase_timestamp','customer_unique_id']].
↪groupby(['customer_unique_id']).max()
# test_2c = part1


#Processing related to "customers"

part2 = customers[['customer_id','customer_unique_id']].
↪groupby(['customer_unique_id']).count()


#Processing related to "payments"

part30 = payments[['payment_value','order_id']].groupby(['order_id']).sum()
order_payment = orders.join(part30, on=['order_id'])
part31 = order_payment[['payment_value', 'customer_id']].
↪merge(customers[['customer_unique_id', 'customer_id']], on='customer_id')

part3 = part31[['payment_value','customer_unique_id']].
↪groupby(['customer_unique_id']).sum()
```

[5]:

```
rfm = (part1.merge(part2, on='customer_unique_id')).merge(part3, on =␣
↪'customer_unique_id')

rfm = rfm.rename(columns={'customer_id':'frequency','order_purchase_timestamp':
↪'recency','payment_value':'monetary'})

rfm['recency']= rfm['recency']-rfm['recency'].max()

rfm
```

[5]:

```
                                            recency  frequency  monetary
customer_unique_id
0000366f3b9a7992bf8c76cfdf3221e2 -112 days +19:55:50          1    141.90
0000b849f77a49e4a4ce2b2a4ca5be3f -115 days +20:10:50          1     27.19
0000f46a3911fa3c0805444483337064 -537 days +06:04:26          1     86.22
0000f6ccb0745a6a4b88665a16c9f078 -321 days +05:29:04          1     43.62
0004aac84e0df4da2b147fca70cf8255 -288 days +04:45:05          1    196.89
…                                                 …         …         …
fffcf5a5ff07b0908bd4e2dbc735a684 -447 days +05:59:59          1   2067.42
fffea47cd6d3cc0a88bd621562a9d061 -262 days +05:07:19          1     84.58
ffff371b4d645b6ecea244b27531430a -568 days +00:48:39          1    112.46
ffff5962728ec6157033ef9805bacc48 -119 days +00:17:04          1    133.69
ffffd2657e2aad2907e67c3e9daecbeb -484 days +05:18:08          1     71.56

[93356 rows x 3 columns]
```

[6]:

```
rfm['recency'].min()
```

[6]:

```
Timedelta('-714 days +21:16:01')
```

Building rfm scores - percentile-based

Function definition

[7]:

```
#Cette fonction fait correspondre à toute valeur d'une variable, un rang qui␣
↪correspond à l'inter-percentile auquel il
#appartient dans la distribution

def my_percentile_rank_fix(v,thr):
    if v <= thr:
        res = 2
    else:
        res = 4
    return res
```

```
#Cette fonction fait correspondre à toute valeur d'une variable, un rang qui␣
↪correspond à l'inter-percentile auquel il
# appartient dans la distribution
def my_percentile_rank(col, v, n):

    range_ = np.linspace(1,n-1,n-1)
    lim=np.percentile(col,range_*(100/n))
    if np.sign(min(v-lim))>0:
        res = np.argmin(abs(v-lim))+2
    else:
        res = np.argmin(abs(v-lim))+1
    return res

#Cette variante de "my_percentile_rank" utilise
def my_percentile_rank_rec(col, v, n):

    range_ = np.linspace(1,n-1,n-1)
    lim=np.percentile(col,range_*(100/n))
    if np.sign(min(v-lim))>np.timedelta64(0,'ns'):# Ici la valeur 0 est de type␣
↪timedelta64
        res = np.argmin(abs(v-lim))+2
    else:
        res = np.argmin(abs(v-lim))+1
    return res

#Cette fonction fait correspondre à toute valeur v d'une variable contenue dans␣
↪col, un rang qui correspond à l'inter-percentile auquel il
#appartient dans la distribution
def my_segment(col, v, n):

    range_ = np.linspace(1,n-1,n-1)
    lim=range_*(mt.floor(col.max()/n))
    if np.sign(min(v-lim))>0:
        res = np.argmin(abs(v-lim))+2
    else:
        res = np.argmin(abs(v-lim))+1
    return res
```

[8]:
```
def my_quintile_recency_rank(v):
    return my_percentile_rank_rec(rfm['recency'],v,5)

def my_quintile_frequency_rank(v):
#    return my_percentile_rank_fix(v,rfm["frequency"].unique().mean())
    return my_percentile_rank_fix(v,1)

def my_quintile_monetary_rank(v):
    return my_percentile_rank(rfm['monetary'],v,5)
```

Actual computation

[9]:
```
#Recency score
start_time = timeit.default_timer()
rfm['recency_s'] = rfm['recency'].apply(lambda x: my_quintile_recency_rank(x))
elapsed = timeit.default_timer() - start_time

print(elapsed)
```

```
905.4680586620016
```

[10]:
```
#Frequency score
start_time = timeit.default_timer()
rfm['frequency_s'] = rfm['frequency'].apply(lambda x:␣
↪my_quintile_frequency_rank(x))
elapsed = timeit.default_timer() - start_time

print(elapsed)
```

```
0.030257911996159237
```

[11]:
```
#Monetary score
start_time = timeit.default_timer()
rfm['monetary_s'] = rfm['monetary'].apply(lambda x:␣
↪my_quintile_monetary_rank(x))
elapsed = timeit.default_timer() - start_time

print(elapsed)
```

```
164.6787575740018
```

[12]:
```
rfm['recency_s'].value_counts()
```

[12]:
```
1    26086
2    19754
3    19359
5    18671
4     9486
Name: recency_s, dtype: int64
```

[13]:
```
# Computation of a synthetic rfm score
rfm['rfm_score']=rfm['recency_s']*rfm['frequency_s']*rfm['monetary_s']
```

[14]:
```
rfm
```

[14]:
```
                                           recency  frequency  monetary  \
customer_unique_id
0000366f3b9a7992bf8c76cfdf3221e2 -112 days +19:55:50          1    141.90
```

```
0000b849f77a49e4a4ce2b2a4ca5be3f -115 days +20:10:50          1     27.19
0000f46a3911fa3c0805444483337064 -537 days +06:04:26          1     86.22
0000f6ccb0745a6a4b88665a16c9f078 -321 days +05:29:04          1     43.62
0004aac84e0df4da2b147fca70cf8255 -288 days +04:45:05          1    196.89
…                                                …        …        …
fffcf5a5ff07b0908bd4e2dbc735a684 -447 days +05:59:59          1   2067.42
fffea47cd6d3cc0a88bd621562a9d061 -262 days +05:07:19          1     84.58
ffff371b4d645b6ecea244b27531430a -568 days +00:48:39          1    112.46
ffff5962728ec6157033ef9805bacc48 -119 days +00:17:04          1    133.69
ffffd2657e2aad2907e67c3e9daecbeb -484 days +05:18:08          1     71.56

                                  recency_s  frequency_s  monetary_s  \
customer_unique_id
0000366f3b9a7992bf8c76cfdf3221e2          4            2           3
0000b849f77a49e4a4ce2b2a4ca5be3f          4            2           1
0000f46a3911fa3c0805444483337064          1            2           2
0000f6ccb0745a6a4b88665a16c9f078          2            2           1
0004aac84e0df4da2b147fca70cf8255          2            2           4
…                                       …          …         …
fffcf5a5ff07b0908bd4e2dbc735a684          1            2           5
fffea47cd6d3cc0a88bd621562a9d061          2            2           2
ffff371b4d645b6ecea244b27531430a          1            2           3
ffff5962728ec6157033ef9805bacc48          4            2           3
ffffd2657e2aad2907e67c3e9daecbeb          1            2           2

                                  rfm_score
customer_unique_id
0000366f3b9a7992bf8c76cfdf3221e2         24
0000b849f77a49e4a4ce2b2a4ca5be3f          8
0000f46a3911fa3c0805444483337064          4
0000f6ccb0745a6a4b88665a16c9f078          4
0004aac84e0df4da2b147fca70cf8255         16
…                                       …
fffcf5a5ff07b0908bd4e2dbc735a684         10
fffea47cd6d3cc0a88bd621562a9d061          8
ffff371b4d645b6ecea244b27531430a          6
ffff5962728ec6157033ef9805bacc48         24
ffffd2657e2aad2907e67c3e9daecbeb          4

[93356 rows x 7 columns]
```

[ ]:

```
[15]: #rfm segment
start_time = timeit.default_timer()
rfm['segment'] = rfm['rfm_score'].apply(lambda x:␣
↪my_segment(rfm['rfm_score'],x,6))
```

```
elapsed = timeit.default_timer() - start_time

print(elapsed)
```

```
14.382687641998928
```

[16]:
```
rfm['segment'].value_counts()
```

[16]:
```
1    76691
2    11947
3     3596
4      510
6      376
5      236
Name: segment, dtype: int64
```

[17]:
```
fig2 =px.scatter_3d(rfm,
                    x='recency_s',
                    y='frequency_s',
                    z='monetary_s',
                    color='segment',
                   )
fig2.update_traces(marker=dict(size=3))
fig2.show()
```

Definition of the segments

Le *rfm_score* a été défini comme le produit des variables scores de chaque client. Ainsi l'aggrégation étent faite par multiplication, il y a une perte de l'information "spatiale" de chaque facteur. C'est pourquoi l'appréciation issue de ce score ne peut se faire que sur une échelle unique globale.

Pour illustrer cela considérons A et B deux clients. A et B n'ont passé en commandes que des produits haut de gamme. Ils ont donc un score monetary de 5. Mais si A l'a fait plusieurs fois - avec une satisfaction confirmée dans ses review_score, au moment de l'ouverture du magasin en ligne; B a commandé moins de fois, mais très récemment: ce dernier a donc pu être exposé à nos nouvelles collections ainsi qu'à notre dernière campagne markéting particulièrement bien performante considérant l'indicateur des liens de redirection facebook et youtube. Ils ont respectivement A(freq:4, recency:2), et B(freq:2, recency:4). Leur rfm_score égal de 40 exprime un **niveau global du potentiel de chiffre d'affaires**.

Segment 1: **l'étincelle**: rfm_score in [1,16]

Segment 2: **la buchette [d'allumette]**: rfm_score in [17,32]

Segment 3: **le briquet**: rfm_score in [33,48]

Segment 4: **le flambeau**: rfm_score in [49,64]

Segment 5: **le feu de camp**: rfm_score in [65,80]

Segment 6: **le dragon**: rfm_score in [81, 100]

0.0.2 Adding new variables related to the customers

Review score

[18]:
```
part40 = reviews[['order_id','review_score']].groupby(['order_id']).mean()
order_review = orders[['order_id','customer_id']].merge(part40, on='order_id')
part41 = order_review[['customer_id','review_score']].
 ↪merge(customers[['customer_id','customer_unique_id']], on='customer_id')
part4 = part41[['customer_unique_id','review_score']].
 ↪groupby(['customer_unique_id']).mean()
```

Number of purchased products

[19]:
```
part50 = items[['order_id','order_item_id']].groupby(['order_id']).sum()
order_items = orders[['order_id','customer_id']].merge(part50, on='order_id')
part51 = order_items[['customer_id','order_item_id']].
 ↪merge(customers[['customer_id','customer_unique_id']], on='customer_id')
part5 = part51[['customer_unique_id','order_item_id']].
 ↪groupby(['customer_unique_id']).sum()
```

[20]:
```
rfm = (rfm.merge(part4, on='customer_unique_id')).merge(part5, on =␣
 ↪'customer_unique_id')
```

[21]:
```
rfm = rfm.rename(columns={'order_item_id':'num_item', 'review_score':
 ↪'review_s'})
```

[22]:
```
score = rfm[['recency_s','monetary_s','review_s','num_item']]
```

[25]:
```
rfm['recency_'] = rfm['recency'].apply(lambda x: to_seconds(x))
```

0.0.3 PCA run over the score dataframe

[26]:
```
X = rfm[['recency_','monetary','review_s','num_item']]
X_scaled = preprocessing.StandardScaler().fit_transform(X)
s_scaled = pd.DataFrame(X_scaled, columns=X.columns, index=X.index)

#Display
s_scaled.describe()
```

[26]:

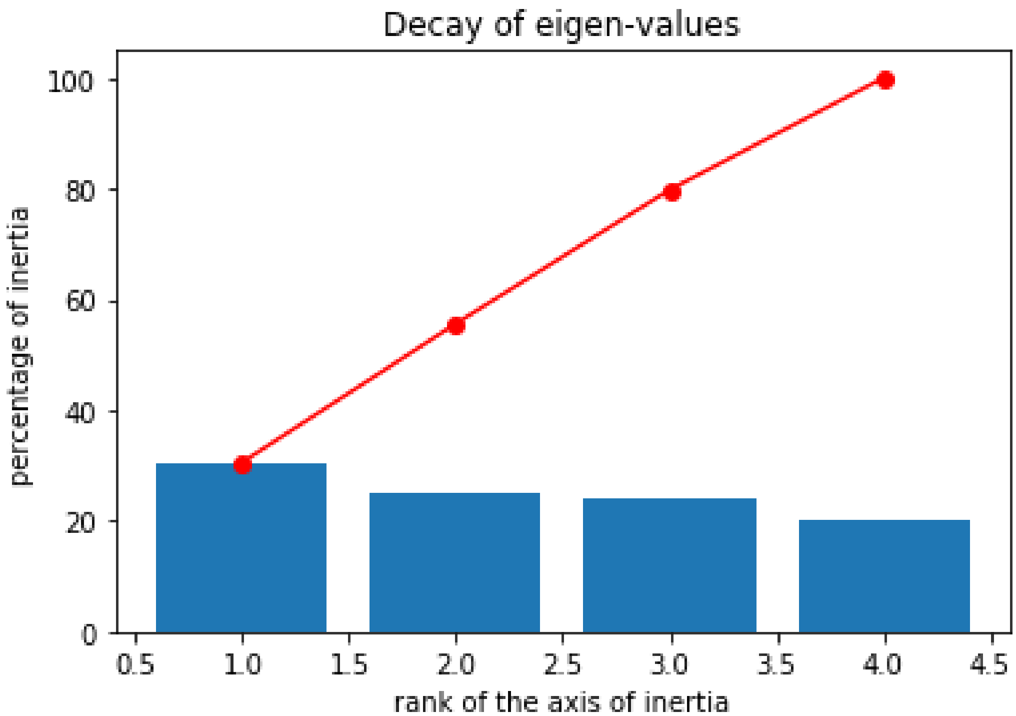
	recency_	monetary	review_s	num_item
count	8.856800e+04	8.856800e+04	8.856800e+04	8.856800e+04
mean	2.058290e-17	1.599656e-16	1.191988e-15	1.156105e-15
std	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00
min	-3.115415e+00	-7.297696e-01	-2.427112e+00	-1.727628e-01
25%	-7.111577e-01	-4.588921e-01	-1.072805e-01	-1.727628e-01
50%	1.235022e-01	-2.520014e-01	6.659967e-01	-1.727628e-01

```
75%    8.097209e-01  8.254519e-02  6.659967e-01 -1.727628e-01
max    1.555779e+00  3.363083e+01  6.659967e-01  9.634392e+01
```

[27]:
```
pca = decomposition.PCA(n_components=4)
n_comp=4
X_projected = pca.fit_transform(s_scaled)
score_pc = pd.DataFrame(X_projected, index=score.index, columns=["F"+str(i+1)␣
 ↪for i in range(n_comp)])

#Display
score_pc
```

[27]:
```
                                        F1        F2        F3        F4
customer_unique_id
0000366f3b9a7992bf8c76cfdf3221e2 -0.405310  0.962735  0.270200  0.038669
0000b849f77a49e4a4ce2b2a4ca5be3f -0.452033  0.670076 -0.588078  0.250455
0000f46a3911fa3c0805444483337064 -0.036028 -2.159198 -0.312470 -0.029606
0000f6ccb0745a6a4b88665a16c9f078 -0.424089 -0.610754 -0.146662  0.217431
0004aac84e0df4da2b147fca70cf8255 -0.263005 -0.113414  0.716821 -0.113463
…                                      …         …         …         …
fffcf5a5ff07b0908bd4e2dbc735a684  5.699430 -0.210566  4.306526 -5.153173
fffea47cd6d3cc0a88bd621562a9d061 -0.300039 -0.224101 -0.195463  0.088717
ffff371b4d645b6ecea244b27531430a -0.533729 -1.899279  1.134881  0.164665
ffff5962728ec6157033ef9805bacc48 -0.429795  0.910205  0.271965  0.064145
ffffd2657e2aad2907e67c3e9daecbeb -0.644042 -1.393117  0.897163  0.281629

[88568 rows x 4 columns]
```

[28]:
```
##Definition de la fonction d'affichage des valeurs propres
def display_scree_plot(pca):
    scree = pca.explained_variance_ratio_*100
    plt.bar(np.arange(len(scree))+1, scree);
    plt.plot(np.arange(len(scree))+1, scree.cumsum(),c="red",marker='o');
    plt.xlabel("rank of the axis of inertia");
    plt.ylabel("percentage of inertia");
    plt.title("Decay of eigen-values");

    plt.savefig('./EigenValuesDecay.png', bbox_inches = 'tight');
    plt.show(block=False);

##Tracé de l'éboulis
display_scree_plot(pca)
```

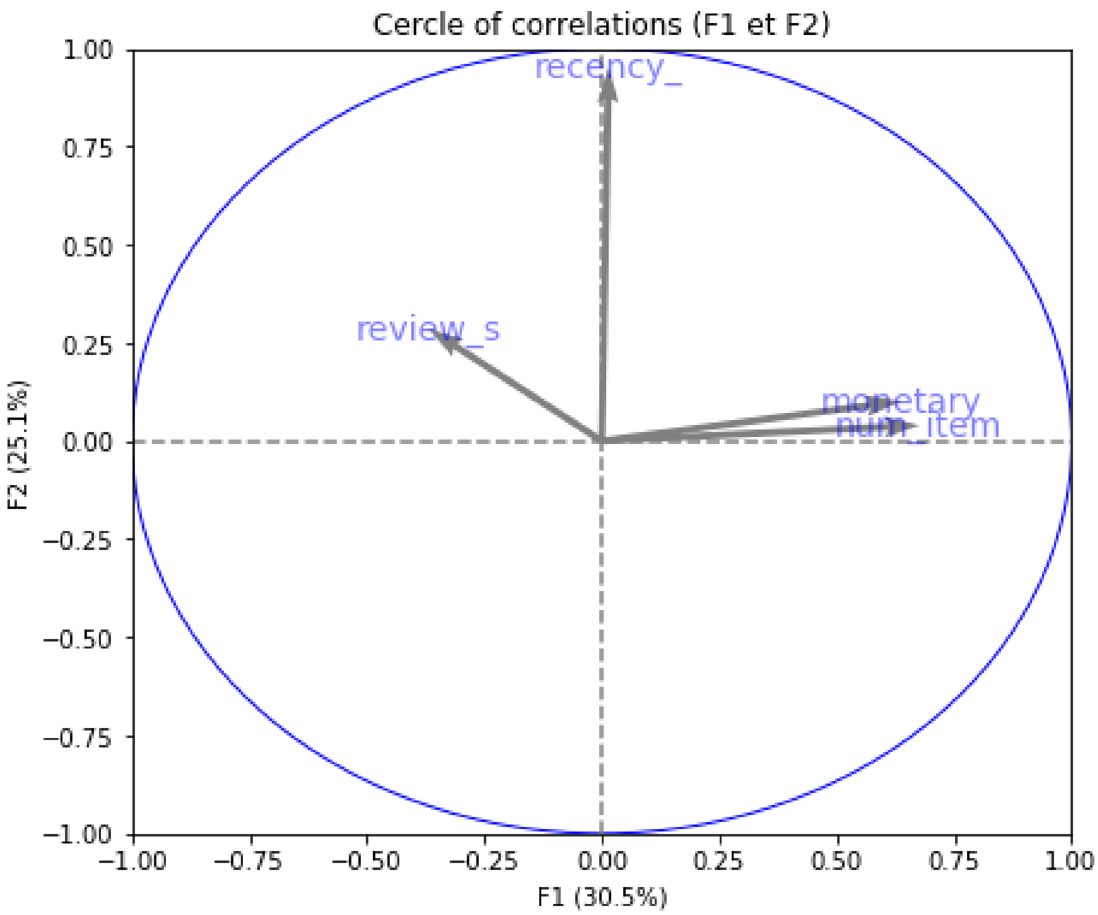


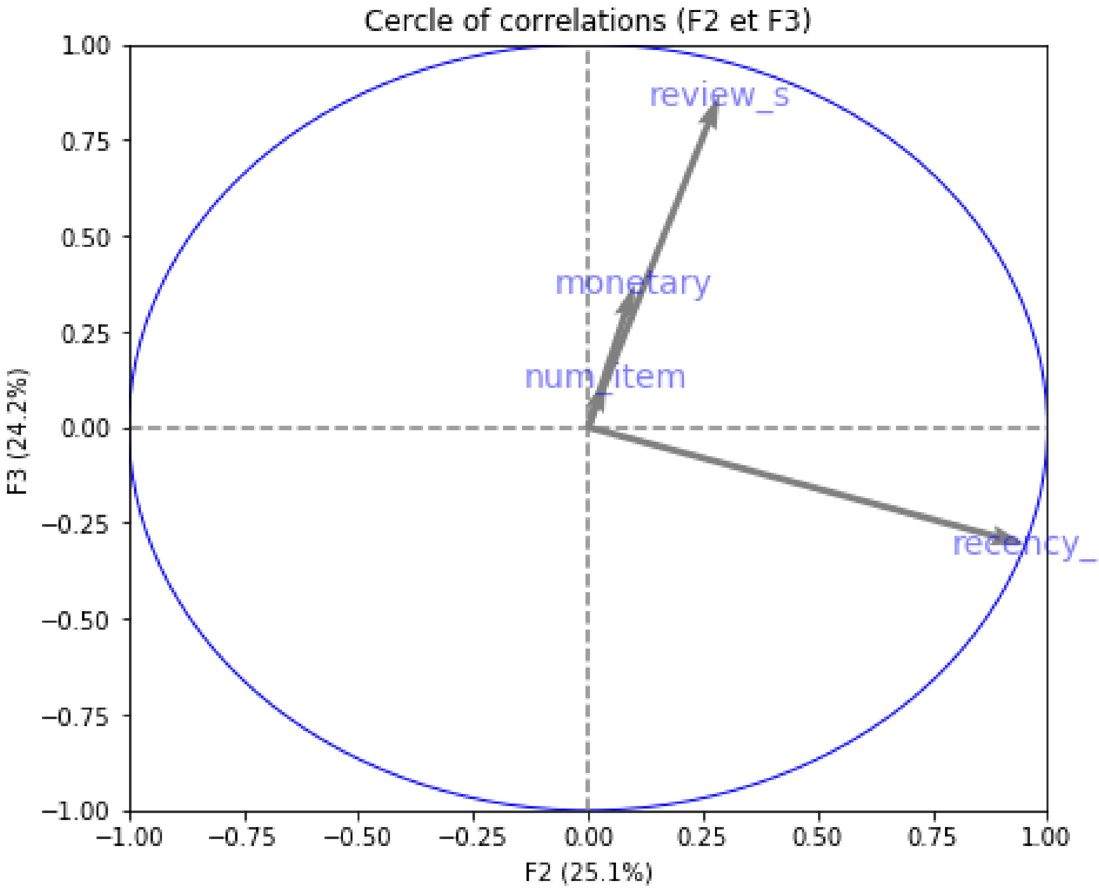
[29]:

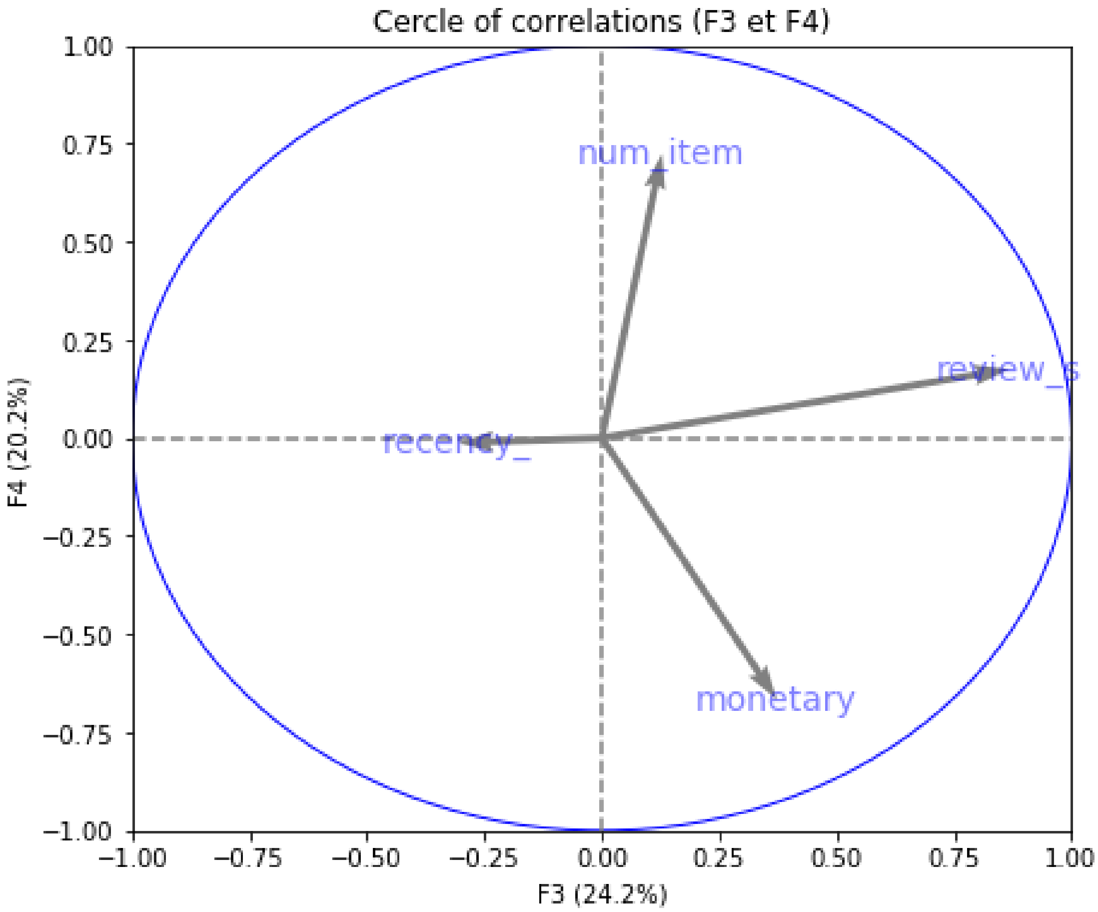
```
##Definition de la fonction d'affichage des valeurs propres
dic_graph={}
def display_circles(pcs, n_comp, pca, axis_ranks, labels=None,␣
↪label_rotation=0, lims=None):
    for d1, d2 in axis_ranks: # On affiche les 3 premiers plans factoriels,␣
↪donc les 6 premières composantes
        if d2 < n_comp:

            # initialisation de la figure
            fig, ax = plt.subplots(figsize=(7,6))

            # détermination des limites du graphique
            if lims is not None :
                xmin, xmax, ymin, ymax = lims
            elif pcs.shape[1] < 30 :
                xmin, xmax, ymin, ymax = -1, 1, -1, 1
            else :
                xmin, xmax, ymin, ymax = min(pcs[d1,:]), max(pcs[d1,:]),␣
↪min(pcs[d2,:]), max(pcs[d2,:])

            # affichage des flèches
            # s'il y a plus de 30 flèches, on n'affiche pas le triangle à leur␣
↪extrémité
```

```
        if pcs.shape[1] < 30 :
            plt.quiver(np.zeros(pcs.shape[1]), np.zeros(pcs.shape[1]),
               pcs[d1,:], pcs[d2,:],
               angles='xy', scale_units='xy', scale=1, color="grey")
            # (voir la doc : https://matplotlib.org/api/_as_gen/matplotlib.
↪pyplot.quiver.html)
        else:
            lines = [[[0,0],[x,y]] for x,y in pcs[[d1,d2]].T]
            ax.add_collection(LineCollection(lines, axes=ax, alpha=.1,␣
↪color='black'))

        # affichage des noms des variables
        if labels is not None:
            for i,(x, y) in enumerate(pcs[[d1,d2]].T):
                if x >= xmin and x <= xmax and y >= ymin and y <= ymax :
                    plt.text(x, y, labels[i], fontsize='14', ha='center',␣
↪va='center', rotation=label_rotation, color="blue", alpha=0.5)

        # affichage du cercle
        circle = plt.Circle((0,0), 1, facecolor='none', edgecolor='b')
        plt.gca().add_artist(circle)

        # définition des limites du graphique
        plt.xlim(xmin, xmax)
        plt.ylim(ymin, ymax)

        # affichage des lignes horizontales et verticales
        plt.plot([-1, 1], [0, 0], color='grey', ls='--')
        plt.plot([0, 0], [-1, 1], color='grey', ls='--')

        # nom des axes, avec le pourcentage d'inertie expliqué
        plt.xlabel('F{} ({}%)'.format(d1+1, round(100*pca.
↪explained_variance_ratio_[d1],1)))
        plt.ylabel('F{} ({}%)'.format(d2+1, round(100*pca.
↪explained_variance_ratio_[d2],1)))

        plt.title("Cercle of correlations (F{} et F{})".format(d1+1, d2+1))

        # Enregistrement du tracé du cercle de corr de ce plan factoriel
        plt.savefig('./CercleCorr'+str(d1)+str(d2)+'.png',␣
↪bbox_inches='tight')


        plt.show(block=False)
```

```
##Tracé du cercle des corrélations
pcs = pca.components_
display_circles(pcs, 4, pca, [(0,1),(1,2),(2,3)], labels=s_scaled.columns)
```




Cercle of correlations (F2 et F3)
review_s
monetary
num_item
recency_
F3 (24.2%)
F2 (25.1%)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

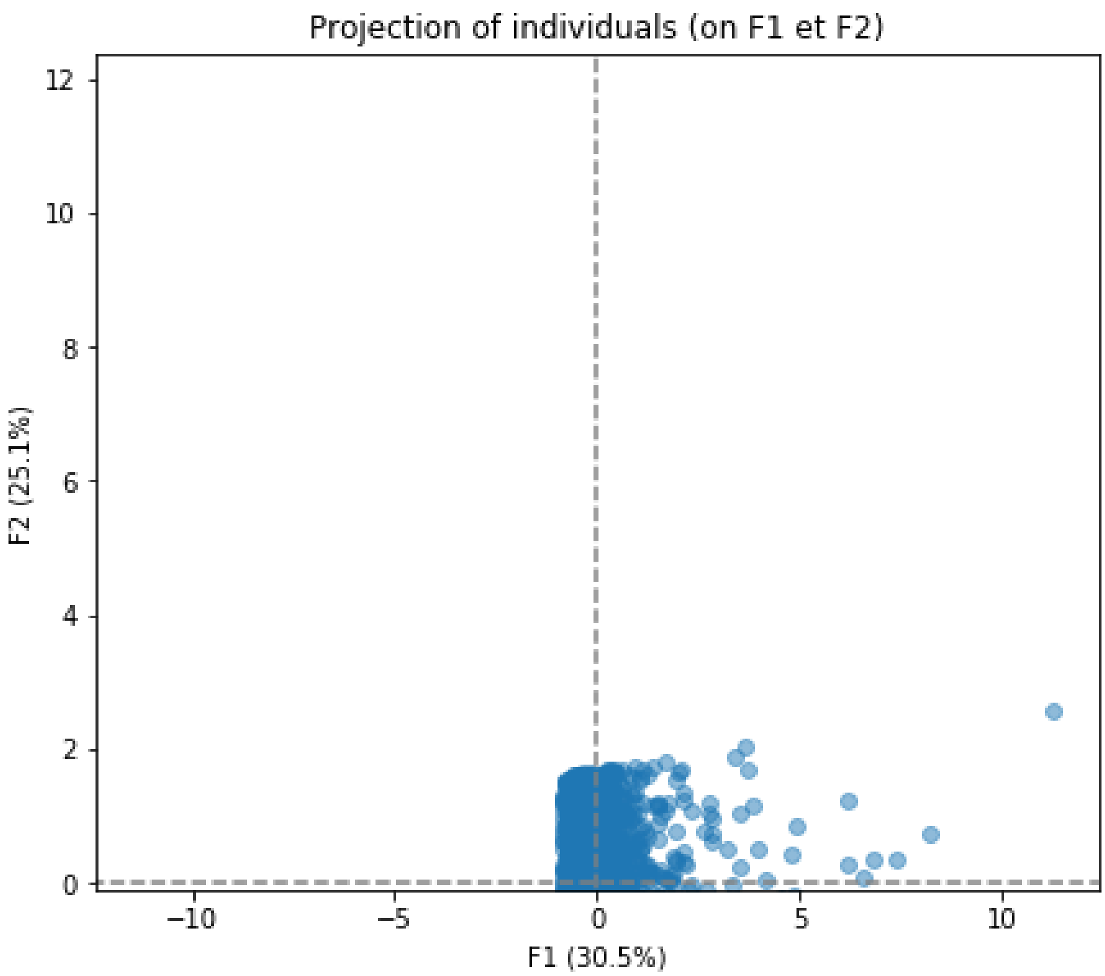


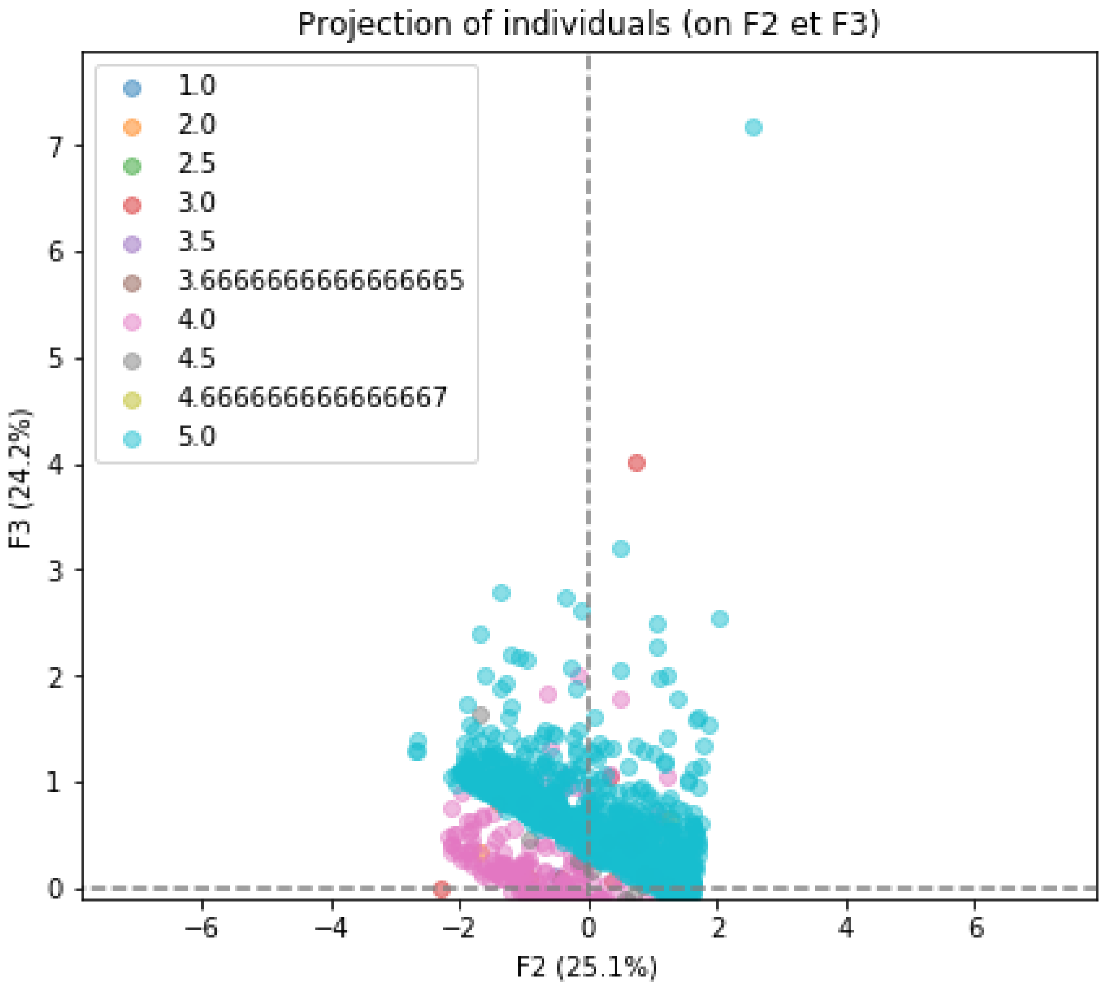
[30]:

```python
##Definition de la fonction d'affichage des valeurs propres
def display_factorial_planes(X_projected, n_comp, pca, axis_ranks, labels=None,␣
↪alpha=0.5, illustrative_var=None):
    for d1,d2 in axis_ranks:
        if d2 < n_comp:

            # initialisation de la figure
            fig = plt.figure(figsize=(7,6))

            # affichage des points
            if illustrative_var is None:
                plt.scatter(X_projected[:, d1], X_projected[:, d2], alpha=alpha)
            else:
                illustrative_var = np.array(illustrative_var)
                for value in np.unique(illustrative_var):
                    selected = np.where(illustrative_var == value)
                    plt.scatter(X_projected[selected, d1],␣
↪X_projected[selected, d2], alpha=alpha, label=value)
```

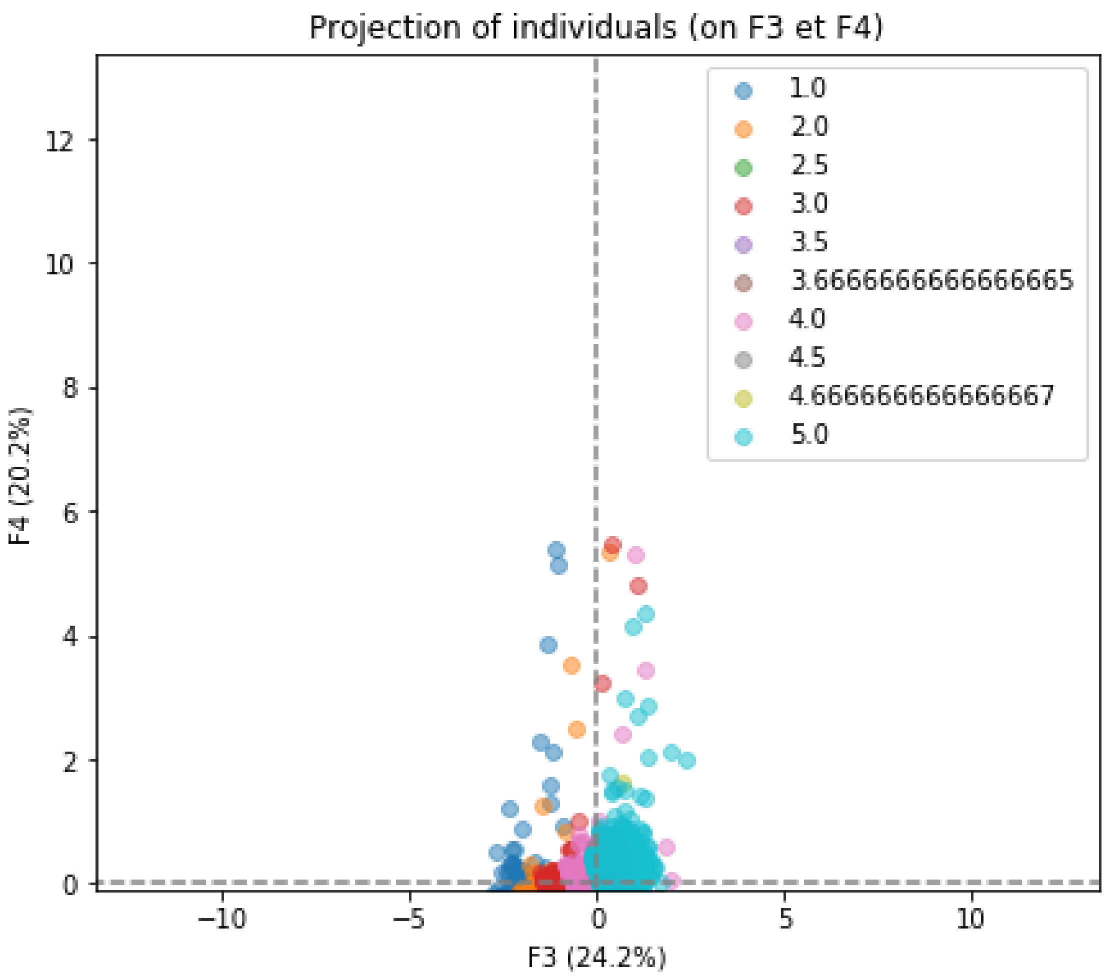
```
                plt.legend()

            # affichage des labels des points
            if labels is not None:
                for i,(x,y) in enumerate(X_projected[:,[d1,d2]]):
                    plt.text(x, y, labels[i],
                              fontsize='14', ha='center',va='center')

            # détermination des limites du graphique
            boundary = np.max(np.abs(X_projected[:, [d1,d2]])) * 1.1
            plt.xlim([-boundary,boundary])
#            plt.ylim([-boundary,boundary])
            plt.ylim([-0.1,boundary])

            # affichage des lignes horizontales et verticales
            plt.plot([-100, 100], [0, 0], color='grey', ls='--')
            plt.plot([0, 0], [-100, 100], color='grey', ls='--')

            # nom des axes, avec le pourcentage d'inertie expliqué
            plt.xlabel('F{} ({}%)'.format(d1+1, round(100*pca.
 ↪explained_variance_ratio_[d1],1)))
            plt.ylabel('F{} ({}%)'.format(d2+1, round(100*pca.
 ↪explained_variance_ratio_[d2],1)))

            plt.title("Projection of individuals (on F{} et F{})".format(d1+1,␣
 ↪d2+1))

            #Enregistrement du tracé
            plt.savefig('./ProjectionOn'+str(d1)+str(d2), bbox_inches='tight')

            plt.show(block=False)


##Tracé du cercle des corrélations
pcs = pca.components_
sampled = np.random.choice(range(X_projected.shape[0]), 2000, replace=False) #!
 ↪rend une liste d'indices

display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(0,1)])
display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(1,2)],␣
 ↪illustrative_var=X['review_s'].iloc[sampled])

#display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(0,1)],␣
 ↪illustrative_var=X['recency_'].iloc[sampled])
#display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(0,1)],␣
 ↪illustrative_var=X['monetary'].iloc[sampled])
```

```
#display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(0,1)],
    illustrative_var=X['review_s'].iloc[sampled])
#display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(0,1)],
    illustrative_var=X['num_item'].iloc[sampled])
```





[31]:
```
display_factorial_planes(X_projected[sampled,:], n_comp, pca, [(2,3)],␣
 ↪illustrative_var=X['review_s'].iloc[sampled])
```



0.0.4 Kmeans based on silouhette score

[32]:
```
#Extraction of the 2 first latent features

latent_features = score_pc[['F2','F3']]
#X = latent_features[sampled,:]
X = latent_features
```

[ ]:
```
```

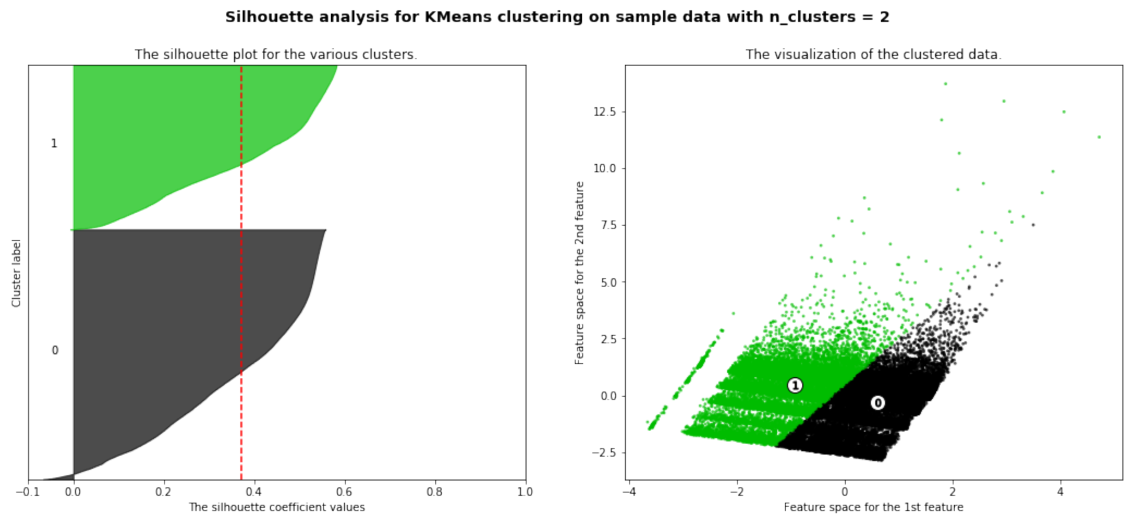
[33]:
```
range_n_clusters = [2, 3, 4, 5, 6]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
```

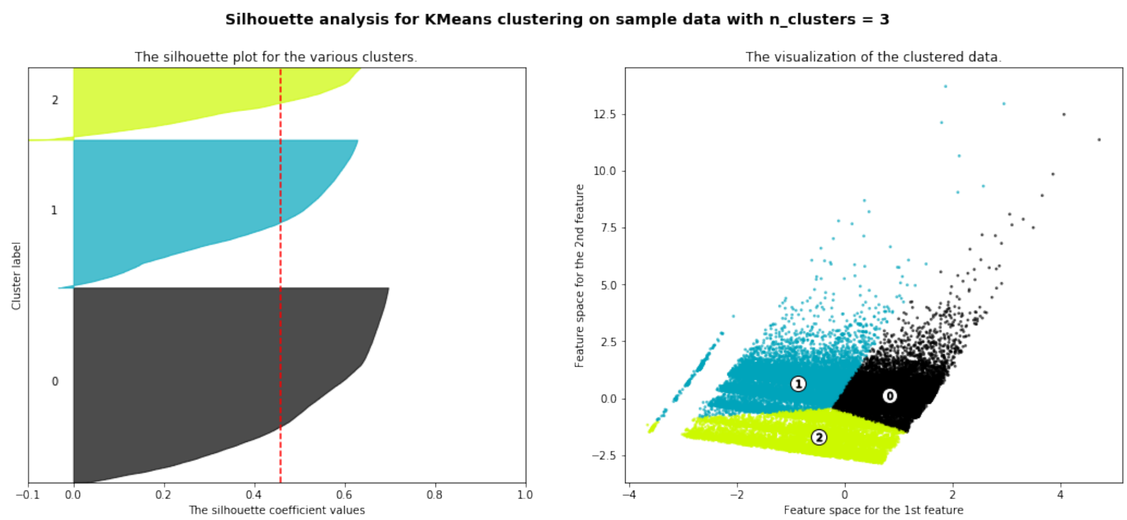
```
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    start_time = timeit.default_timer()
    silhouette_avg = silhouette_score(X, cluster_labels,sample_size=10000)␣
↪#Used argument sample_size on Dec 28, 2020
    elapsed = timeit.default_timer() - start_time
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg, "and its␣
↪computation time is: ", elapsed) # time computation added on Dec 28, 2020

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    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[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
```

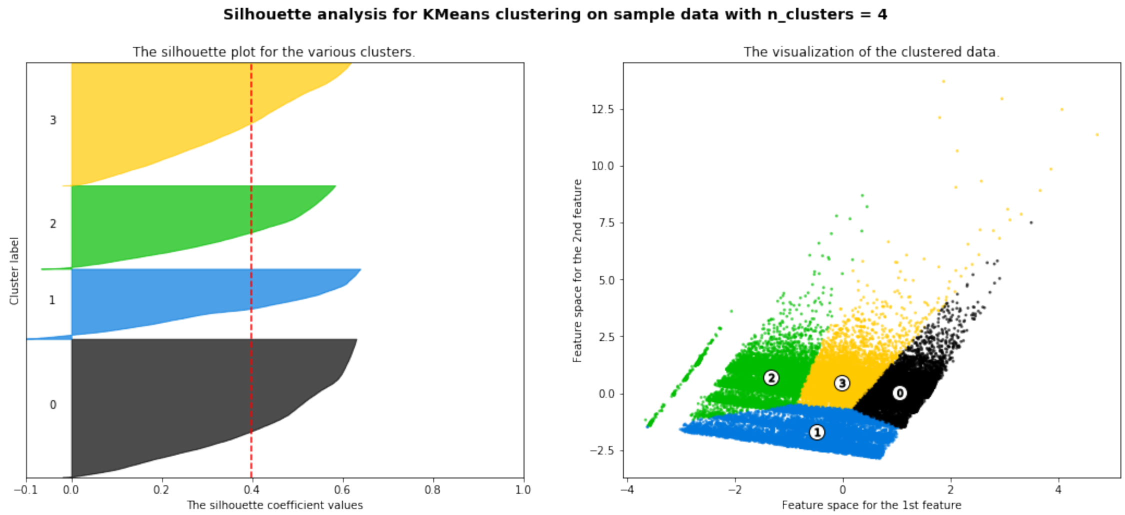
```
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X.iloc[:, 0], X.iloc[:, 1], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

    plt.savefig('./HistoPayValues_%d.png' % n_clusters, bbox_inches='tight');

plt.show()
```

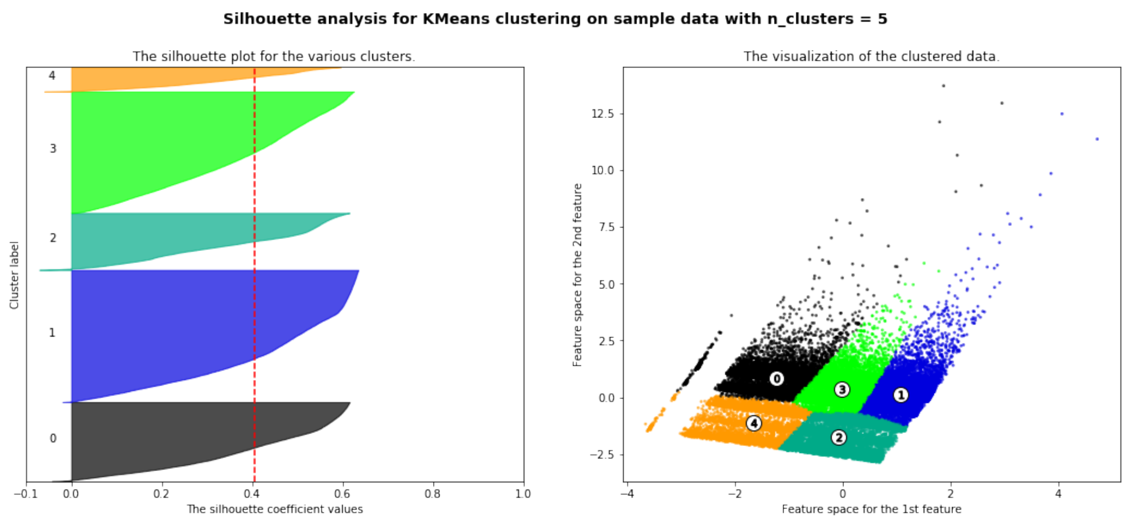
```
For n_clusters = 2 The average silhouette_score is : 0.37268564662444226 and its
computation time is:  0.8707660030049738
For n_clusters = 3 The average silhouette_score is : 0.4588228312073049 and its
computation time is:  0.9076170300031663
For n_clusters = 4 The average silhouette_score is : 0.3982032764823717 and its
```

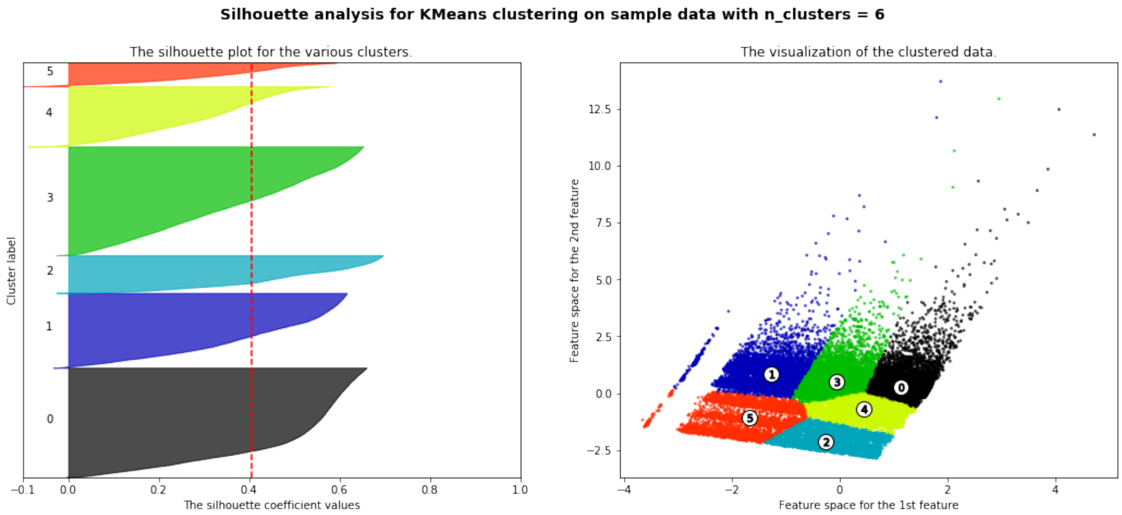
```
computation time is:  0.8008753609974519
For n_clusters = 5 The average silhouette_score is : 0.4054340042179942 and its
computation time is:  0.795828957001504
For n_clusters = 6 The average silhouette_score is : 0.40478601244076334 and its
computation time is:  0.8044797370021115
```






Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature


Silhouette analysis for KMeans clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

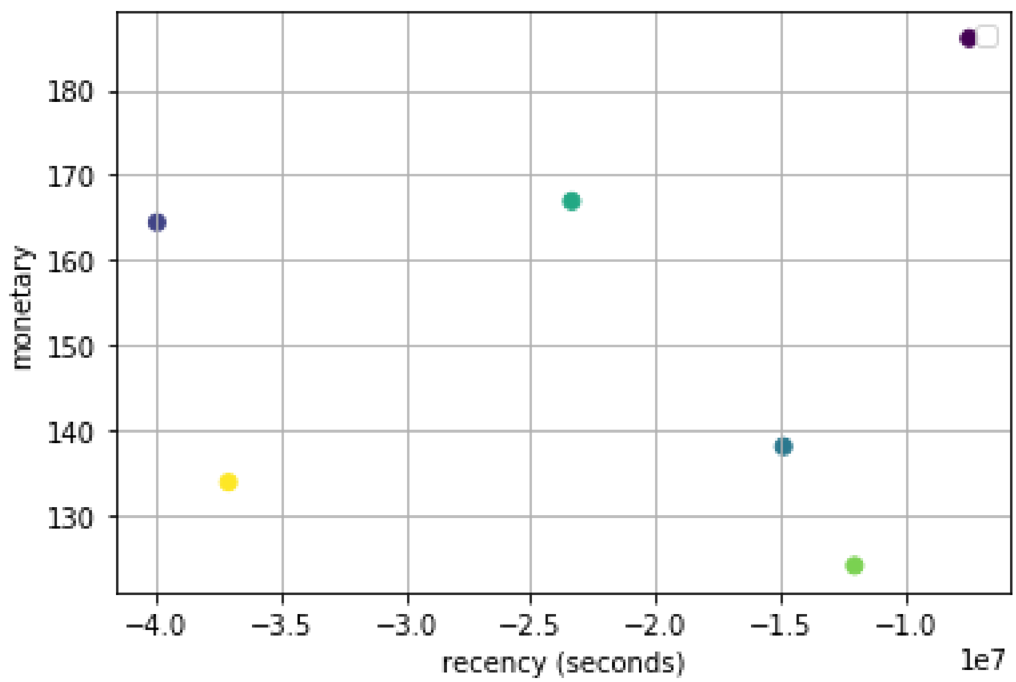


[34]:
```
km_all = rfm[['recency_','monetary','review_s','num_item']]
#km_all = latent_features
km_all['segment']=cluster_labels

km_seg_counts = km_all.groupby(['segment']).count()
km_seg_mean = km_all.groupby(['segment']).mean()
km_seg_var = km_all.groupby(['segment']).var()
```

```
/home/erbadi/anaconda3/lib/python3.7/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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[35]:
```
plt.scatter(km_seg_mean['recency_'],km_seg_mean['monetary'], c=km_seg_mean.
 ↪index);
plt.xlabel('recency (seconds)')
plt.ylabel('monetary')
plt.legend()
plt.grid()

plt.savefig('./Centroids', bbox_inches='tight')
```

```
No handles with labels found to put in legend.
```



[36]:
```
km_seg_mean
```

[36]:

segment	recency_	monetary	review_s	num_item
0	-7.476280e+06	186.035972	4.919927	1.473606
1	-3.992703e+07	164.432451	4.703270	1.347842
2	-1.489350e+07	138.111701	1.193843	1.522264
3	-2.335253e+07	166.913704	4.775549	1.389434
4	-1.205166e+07	124.132501	3.529963	1.331907
5	-3.707289e+07	133.917814	2.048375	1.455801

Segment 0: le dragon déchainé

Segment 1: le phoenix endormi

Segment 2: la braise fraîche

Segment 3: le feu de camp interrompu

Segment 4: l'étincelle

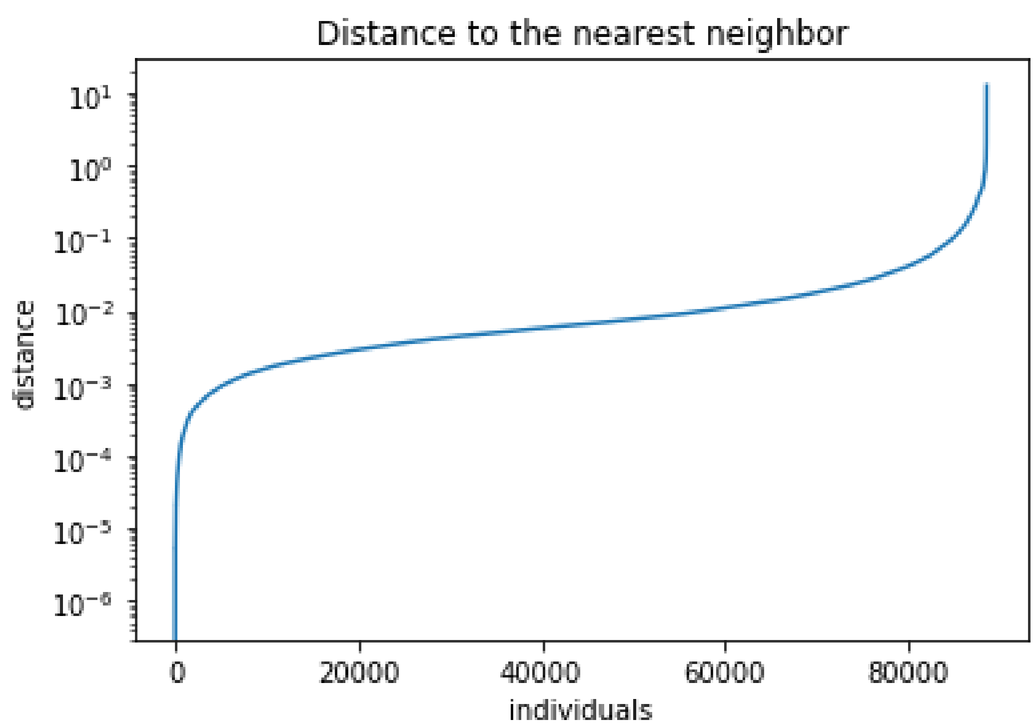
Segment 5: l'allumette fumante

0.0.5 DBScan segmentation

Determination of the optimal epsilon

[37]:

```python
%matplotlib inline
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import numpy as np

neigh = NearestNeighbors(n_neighbors=5)
nbrs = neigh.fit(score_pc)
distances, indices = nbrs.kneighbors(score_pc)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances);
plt.semilogy()
plt.xlabel('individuals')
plt.ylabel('distance')
plt.title('Distance to the nearest neighbor')

plt.savefig('./NearestNeighbor.png', bbox_inches='tight')
```

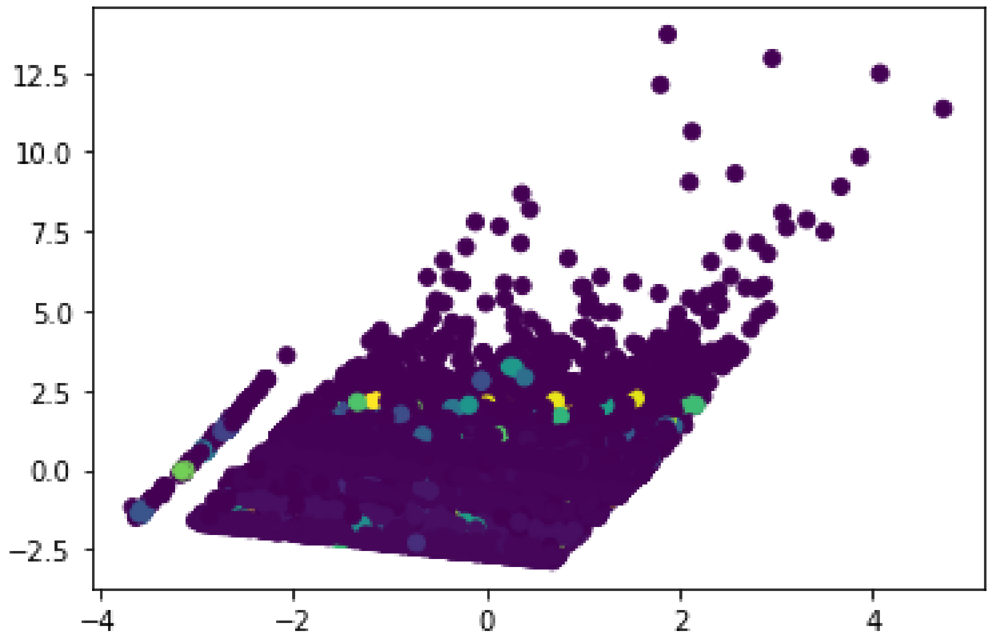


Une bonne valeur de epsilon serait donc **0.1**

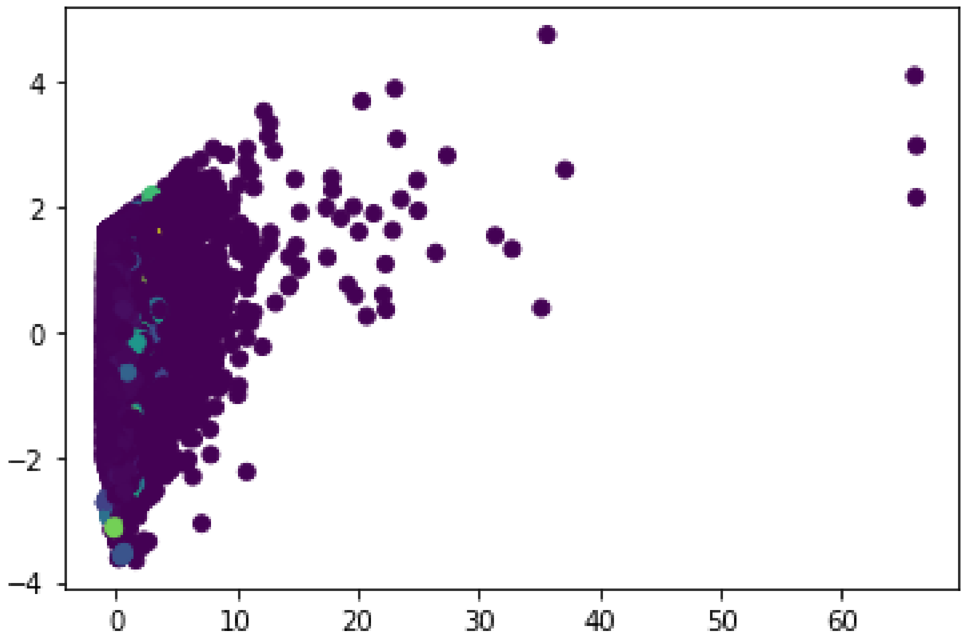
Segmentation

[38]:
```
from sklearn.cluster import DBSCAN

start_time = timeit.default_timer()
y_pred = DBSCAN(eps = 0.1, min_samples=5).fit_predict(score_pc)
elapsed = timeit.default_timer() - start_time
print(elapsed)

plt.scatter(score_pc['F2'],score_pc['F3'],c = y_pred);
```

```
1.7405096310030785
```



[39]:
```
plt.scatter(score_pc['F1'],score_pc['F2'],c = y_pred);
```

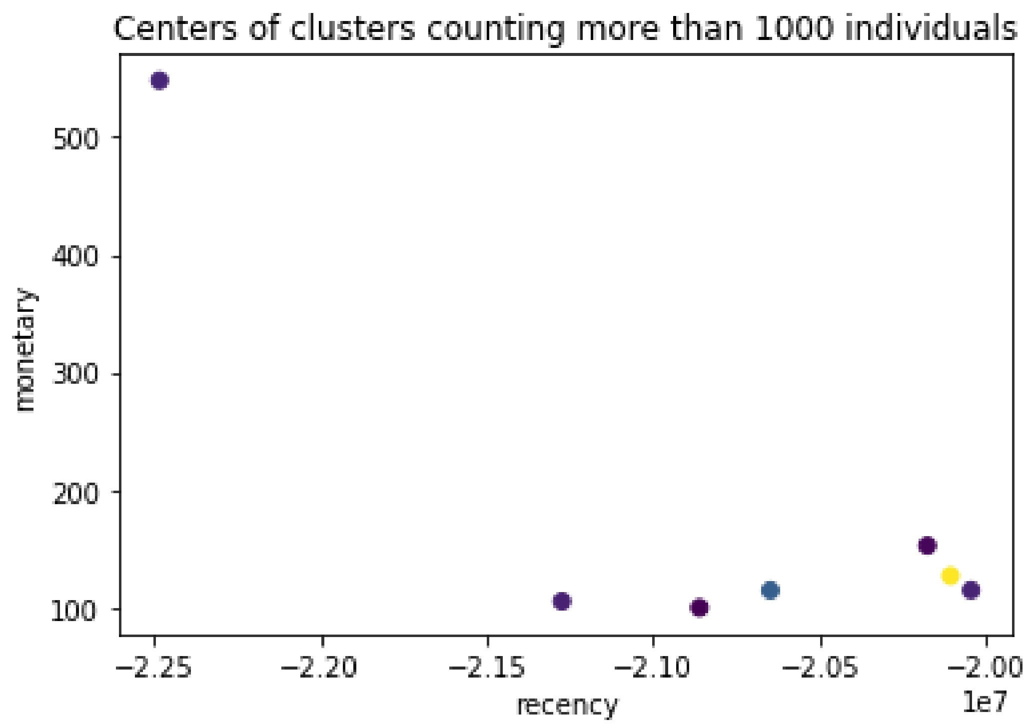


Interpretation

[40]:
```
#Extracting the means and sqrt of each segment
#db_all = score_pc[['F2','F3']]
db_all = rfm[['recency_','monetary','review_s','num_item']]
db_all['segment']=y_pred

seg_counts = db_all.groupby(['segment']).count()
seg_mean = db_all.groupby(['segment']).mean()
seg_sqrt = db_all.groupby(['segment']).var()

thresh = 1000
populated_seg = seg_mean.where(seg_counts['recency_']>=thresh)
#db_all.groupby(['segment']).mean()
plt.scatter(populated_seg['recency_'],populated_seg['monetary'],␣
 ↪c=seg_counts['recency_']);
plt.xlabel('recency')
plt.ylabel('monetary')
plt.title('Centers of clusters counting more than %d individuals' % thresh )

plt.savefig('./Clusters_%d.png' % thresh, bbox_inches='tight');
print('Le nombre de clients couverts par les segments ayant un effectif de plus␣
 ↪de: ',thresh, ' est: ',␣
 ↪seg_counts[seg_counts['recency_']>=thresh]['recency_'].sum())
```

```
Le nombre de clients couverts par les segments ayant un effectif de plus de:
```

```
1000  est:  83911

/home/erbadi/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4:
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: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```



0.0.6 Segment stability

New version

[41]:
```python
# Function which updates the period considered and then computes the different␣
↪aggregated variables used for the subsequent segmentation
def extended_basis(order, b, timestep, init='recency'):
    #Updating the period covered
    part0 = order[order['order_purchase_timestamp'] <= b[init].max() + datetime.
↪timedelta(weeks=+timestep)]
```

```
    #Processing related to "orders"
    order_custom = part0.merge(customers[['customer_id',␣
↪'customer_unique_id']], on='customer_id')
    part1 = order_custom[['order_purchase_timestamp','customer_unique_id']].
↪groupby(['customer_unique_id']).max()

    #Processing related to "customers"
#    part2 = customers[['customer_id','customer_unique_id']].
↪groupby(['customer_unique_id']).count()

    #Processing related to "payments"
    part30 = payments[['payment_value','order_id']].groupby(['order_id']).sum()
    order_payment = part0.join(part30, on=['order_id'])
    part31 = order_payment[['payment_value', 'customer_id']].
↪merge(customers[['customer_unique_id', 'customer_id']], on='customer_id')
    part3 = part31[['payment_value','customer_unique_id']].
↪groupby(['customer_unique_id']).sum()

    #Processing related to reviews
    part40 = reviews[['order_id','review_score']].groupby(['order_id']).mean()
    order_review = part0[['order_id','customer_id']].merge(part40,␣
↪on='order_id')
    part41 = order_review[['customer_id','review_score']].
↪merge(customers[['customer_id','customer_unique_id']], on='customer_id')
    part4 = part41[['customer_unique_id','review_score']].
↪groupby(['customer_unique_id']).mean()

    #Processing related to items [purchased]
    part50 = items[['order_id','order_item_id']].groupby(['order_id']).sum()
    order_items = part0[['order_id','customer_id']].merge(part50, on='order_id')
    part51 = order_items[['customer_id','order_item_id']].
↪merge(customers[['customer_id','customer_unique_id']], on='customer_id')
    part5 = part51[['customer_unique_id','order_item_id']].
↪groupby(['customer_unique_id']).sum()

    rfm = ((part1.merge(part3, on='customer_unique_id')).merge(part4, on =␣
↪'customer_unique_id')).merge(part5, on = 'customer_unique_id')
    rfm = rfm.rename(columns={'order_purchase_timestamp':
↪'recency','payment_value':'monetary','order_item_id':'num_item',␣
↪'review_score':'review_s'})
    rfm['recency']= rfm['recency']-order['order_purchase_timestamp'].max()
    res = rfm

    return res
```

```
# Function used to compute a sclaing of the data, in order for them to match␣
↪with the '.fit(X)' arguments requirements
def scaled_basis(b, fitted_scaler):
    X = b[['monetary','review_s','num_item']]
    X['recency_'] = b['recency'].apply(lambda x: to_seconds(x))
    X_scaled = fitted_scaler.transform(X)
    b_scaled = pd.DataFrame(X_scaled, columns=X.columns, index=X.index)
    return b_scaled
```

[42]:

```
def stability_check(data, b0, n_clusters, timestep, num_timestep):

    #Building and fitting the scaler of the bases
    X = b0[['monetary','review_s','num_item']]
    X['recency_'] = b0['recency'].apply(lambda x: to_seconds(x))
    fitted_scaler = preprocessing.StandardScaler().fit(X)

    #Building and fitting the clusterer
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    c0 = clusterer.fit(scaled_basis(b0, fitted_scaler))


    bases = {'b0':b0}
    clusterers = {'c0':c0}
    current_clusterings = {}
    updated_clusterings = {}
    rand_scores = {}
    for ind in range(num_timestep):
        #Future bases and forecast clusterings (with current clusterer)
        bases['b'+str(ind+1)] =␣
↪extended_basis(data,bases['b'+str(ind)],timestep)
        current_clusterings['c0.'+str(ind+1)] = c0.
↪predict(scaled_basis(bases['b'+str(ind+1)], fitted_scaler))

        #Updated clusterings (-ers)
        clusterer = KMeans(n_clusters=n_clusters, random_state=10)
        clusterers['c'+str(ind+1)] = clusterer.
↪fit(scaled_basis(bases['b'+str(ind+1)], fitted_scaler))
        updated_clusterings['c'+str(ind+1)+'.'+str(ind+1)] =␣
↪clusterers['c'+str(ind+1)].predict(scaled_basis(bases['b'+str(ind+1)],␣
↪fitted_scaler))

        #Rand score computation
        rand_scores[ind+1] = adjusted_rand_score(current_clusterings['c0.
↪'+str(ind+1)],
                                                                        ␣
↪updated_clusterings['c'+str(ind+1)+'.'+str(ind+1)])
```

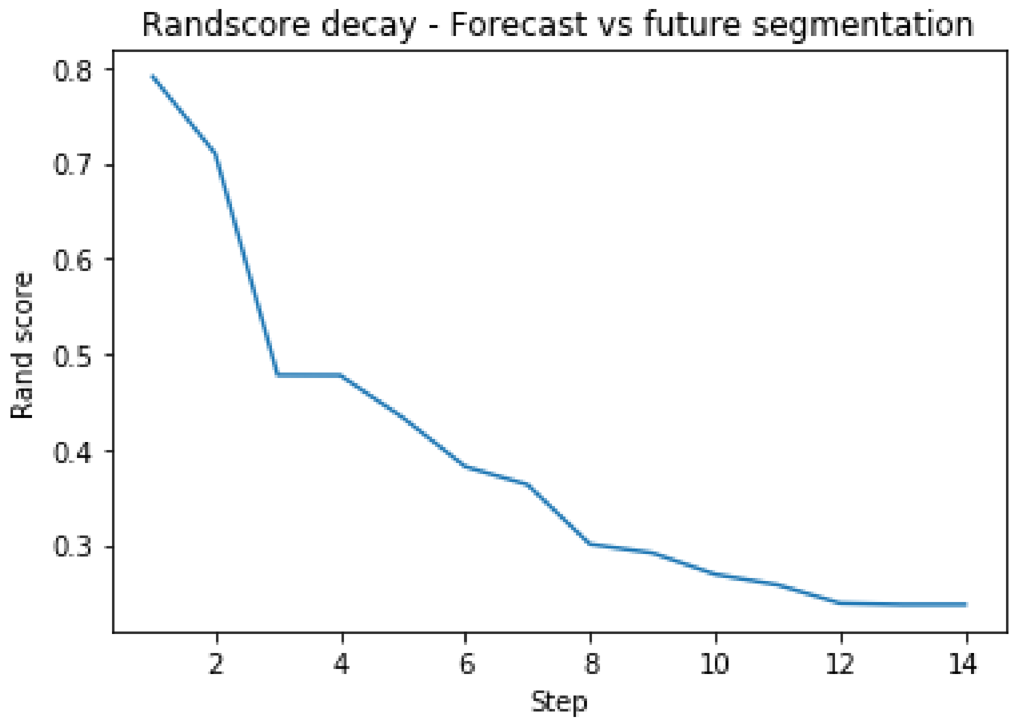
```
    #Plotting the rand score evolution
    lists = sorted(rand_scores.items()) # sorted by key, return a list of tuples
    x, y = zip(*lists) # unpack a list of pairs into two tuples
    plt.plot(x, y)
    plt.xlabel('Step')
    plt.ylabel('Rand score')
    plt.title('Randscore decay - Forecast vs future segmentation')

    plt.savefig('./Ebouli_Randscore.png', bbox_inches='tight')
    plt.show()

    #return the rand score
    return bases
```

Actual computation

[43]:
```
#Adjustment of the dataset orders to the needs of the computation -> from␣
↪datetime to timedelta by recentering
orders['order_purchase_timestamp'] = orders['order_purchase_timestamp'] -␣
↪orders['order_purchase_timestamp'].max()
```

[44]:
```
#Parameters for the computation of the segments stability assessment
data = orders
init_time = orders['order_purchase_timestamp'].min()
init_b = orders[orders['order_purchase_timestamp']== init_time]

b0 = extended_basis(orders, init_b, 52, init='order_purchase_timestamp')
#b0 = data[data['recency'] <= data['recency'].min() + datetime.
↪timedelta(weeks=+52)]
n_clusters = 6
timestep = 4 #time step in weeks unit
num_timestep = 14
```

[45]:
```
start_time = timeit.default_timer()
res = stability_check(data, b0, n_clusters, timestep, num_timestep)
elapsed = timeit.default_timer() - start_time

print('The computation time is: ', elapsed)
```




```
The computation time is:  40.18205604499963
```

[46]:
```python
for ind in list(res.values()):
    print(ind.shape)
```

```
(22337, 4)
(25883, 4)
(29548, 4)
(36680, 4)
(40760, 4)
(46771, 4)
(52748, 4)
(58681, 4)
(64428, 4)
(70660, 4)
(75284, 4)
(79927, 4)
(86689, 4)
(88568, 4)
(88568, 4)
```

1 ssssssssssssssssssssssssssssssssssssssssss

[ ]:

[47]:
```
b0
```

[47]:
```
                                           recency  monetary  review_s  \
customer_unique_id
0000f46a3911fa3c0805444483337064 -537 days +06:04:26     86.22       3.0
0005e1862207bf6ccc02e4228effd9a0 -543 days +08:31:35    150.12       4.0
0006fdc98a402fceb4eb0ee528f6a8d4 -408 days +18:22:33     29.00       3.0
000a5ad9c4601d2bbdd9ed765d5213b3 -384 days +22:44:38     91.28       4.0
000de6019bb59f34c099a907c151d855 -377 days +04:09:56    257.44       2.0
…                                                …         …         …
fff3a9369e4b7102fab406a334a678c3 -384 days +19:26:01    102.74       5.0
fff699c184bcc967d62fa2c6171765f7 -362 days +02:06:17     55.00       4.0
fffcf5a5ff07b0908bd4e2dbc735a684 -447 days +05:59:59   2067.42       5.0
ffff371b4d645b6ecea244b27531430a -568 days +00:48:39    112.46       5.0
ffffd2657e2aad2907e67c3e9daecbeb -484 days +05:18:08     71.56       5.0

                                  num_item
customer_unique_id
0000f46a3911fa3c0805444483337064         1
0005e1862207bf6ccc02e4228effd9a0         1
0006fdc98a402fceb4eb0ee528f6a8d4         1
000a5ad9c4601d2bbdd9ed765d5213b3         1
000de6019bb59f34c099a907c151d855         3
…                                      …
fff3a9369e4b7102fab406a334a678c3         1
fff699c184bcc967d62fa2c6171765f7         1
fffcf5a5ff07b0908bd4e2dbc735a684         3
ffff371b4d645b6ecea244b27531430a         1
ffffd2657e2aad2907e67c3e9daecbeb         1

[22337 rows x 4 columns]
```