In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import datetime, nltk, warnings
import matplotlib.cm as cm
import itertools
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn import preprocessing, model_selection, metrics, feature_selection
from sklearn.model_selection import GridSearchCV, learning_curve
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn import neighbors, linear_model, svm, tree, ensemble
from sklearn.ensemble import AdaBoostClassifier
from sklearn.decomposition import PCA
from IPython.display import display, HTML
#import plotly.plotly as py
#import plotly.graph_objs as go
#from plotly.offline import init_notebook_mode,iplot
#init_notebook_mode(connected=True)
#warnings.filterwarnings("ignore")
#plt.rcParams["patch.force_edgecolor"] = True
#plt.style.use('fivethirtyeight')
#mpl.rc('patch', edgecolor = 'dimgray', linewidth=1)
%matplotlib inline
```

** Data Preparation **

In [2]:

```
# read the datafile
df_initial = pd.read_csv('data.csv',encoding="ISO-8859-1",
                         dtype={'CustomerID': str,'InvoiceID': str})
print('Dataframe dimensions:', df_initial.shape)
df_initial['InvoiceDate'] = pd.to_datetime(df_initial['InvoiceDate'])
```

```
Dataframe dimensions: (541909, 8)
```

In [3]:

```
# show first lines
display(df_initial[:5])
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850	United Kingdom

** Exploratory Data Analysis **

Identify null values

In [4]:

```
# gives some infomation on columns types and number of null values
tab_info=pd.DataFrame(df_initial.dtypes).T.rename(index={0:'column type'})
tab_info=tab_info.append(pd.DataFrame(df_initial.isnull().sum()).T.rename(index={0:'null va
tab_info=tab_info.append(pd.DataFrame(df_initial.isnull().sum()/df_initial.shape[0]*100).T.
                         rename(index={0:'null values (%)'}))
print ('-' * 10 + " Display information about column types and number of null values " + '-
print
display(tab_info)
```

```
---------- Display information about column types and number of null values
----------
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
column type	object	object	object	int64	datetime64[ns]	float64	object	
null values (nb)	0	0	1454	0	0	0	135080	
null values (%)	0	0	0.268311	0	0	0	24.9267	

In [5]:

```
df_initial.dropna(axis = 0, subset = ['CustomerID'], inplace = True)
print('Dataframe dimensions:', df_initial.shape)
# gives some information on columns types and number of null values
tab_info=pd.DataFrame(df_initial.dtypes).T.rename(index={0:'column type'})
tab_info=tab_info.append(pd.DataFrame(df_initial.isnull().sum()).T.rename(index={0:'null va
tab_info=tab_info.append(pd.DataFrame(df_initial.isnull().sum()/df_initial.shape[0]*100).T.
                         rename(index={0:'null values (%)'}))
display(tab_info)
```

```
Dataframe dimensions: (406829, 8)
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Co
column type	object	object	object	int64	datetime64[ns]	float64	object	
null values (nb)	0	0	0	0	0	0	0	
null values (%)	0	0	0	0	0	0	0	

In [6]:

```
print('Duplicate data entries: {}'.format(df_initial.duplicated().sum()))
df_initial.drop_duplicates(inplace = True)
```

```
Duplicate data entries: 5225
```

Exploring data attributes

In [7]:

```
temp = df_initial[['CustomerID', 'InvoiceNo', 'Country']].groupby(
    ['CustomerID', 'InvoiceNo', 'Country']).count()
temp = temp.reset_index(drop = False)
countries = temp['Country'].value_counts()
print('No. of cuntries in dataframe: {}'.format(len(countries)))
```

```
No. of cuntries in dataframe: 37
```

In [8]:

```
temp_no_of_order_per_count = df_initial[['CustomerID','Country']].groupby(['Country']).coun
temp_no_of_order_per_count = temp_no_of_order_per_count.reset_index(drop = False)

print('-' * 10 + " Contry-wise order calculation "+ '-' * 10)
print
print (temp_no_of_order_per_count.sort_values(
    by='CustomerID', ascending=False).rename(index=str,
                                        columns={"CustomerID": "Country wise number of orde
```

```
---------- Contry-wise order calculation ----------
                Country  Country wise number of order
35        United Kingdom                        356728
14               Germany                          9480
13                France                          8475
10                  EIRE                          7475
30                 Spain                          2528
23           Netherlands                          2371
3                Belgium                          2069
32           Switzerland                          1877
26              Portugal                          1471
0               Australia                          1258
24                Norway                          1086
18                 Italy                           803
6         Channel Islands                           757
12                Finland                           695
7                  Cyprus                           611
31                 Sweden                           461
1                 Austria                           401
9                 Denmark                           389
19                  Japan                           358
25                 Poland                           341
33                    USA                           291
17                 Israel                           247
36           Unspecified                           241
29             Singapore                           229
16               Iceland                           182
5                 Canada                           151
15                Greece                           146
22                 Malta                           127
34  United Arab Emirates                            68
11    European Community                            61
27                   RSA                            58
20               Lebanon                            45
21             Lithuania                            35
4                 Brazil                            32
8          Czech Republic                            30
2                 Bahrain                            17
28          Saudi Arabia                            10
```

In [9]:

```
pd.DataFrame([{'products': len(df_initial['StockCode'].value_counts()),
               'transactions': len(df_initial['InvoiceNo'].value_counts()),
               'customers': len(df_initial['CustomerID'].value_counts()),
              }], columns = ['products', 'transactions', 'customers'],
              index = ['quantity'])
```

Out[9]:

	products	transactions	customers
quantity	3684	22190	4372

In [10]:

```
temp = df_initial.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate'].co
nb_products_per_basket = temp.rename(columns = {'InvoiceDate':'Number of products'})
nb_products_per_basket[:10].sort_values('CustomerID')
```

Out[10]:

	CustomerID	InvoiceNo	Number of products
0	12346	541431	1
1	12346	C541433	1
2	12347	537626	31
3	12347	542237	29
4	12347	549222	24
5	12347	556201	18
6	12347	562032	22
7	12347	573511	47
8	12347	581180	11
9	12348	539318	17

In [11]:

```
nb_products_per_basket['order_cancelled'] = nb_products_per_basket['InvoiceNo'].apply(
    lambda x:int('C' in x))
display(nb_products_per_basket[:5])


n1 = nb_products_per_basket['order_cancelled'].sum()
n2 = nb_products_per_basket.shape[0]
percentage = (n1/n2)*100
print('Number of orders cancelled: {}/{} ({:.2f}%) '.format(n1, n2, percentage))
```

Out[ ]:

	CustomerID	InvoiceNo	Number of products	order_cancelled
0	12346	541431	1	0
1	12346	C541433	1	1
2	12347	537626	31	0
3	12347	542237	29	0
4	12347	549222	24	0

```
Number of orders cancelled: 3654/22190 (16.47%)
```

In [12]:

```
display(df_initial.sort_values('CustomerID')[:5])
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011-01-18 10:01:00	1.04	12346	U King
61624	C541433	23166	MEDIUM CERAMIC TOP STORAGE JAR	-74215	2011-01-18 10:17:00	1.04	12346	U King
286623	562032	22375	AIRLINE BAG VINTAGE JET SET BROWN	4	2011-08-02 08:48:00	4.25	12347	Ice
72260	542237	84991	60 TEATIME FAIRY CAKE CASES	24	2011-01-26 14:30:00	0.55	12347	Ice
14943	537626	22772	PINK DRAWER KNOB ACRYLIC EDWARDIAN	12	2010-12-07 14:57:00	1.25	12347	Ice

In [13]:

```
df_check = df_initial[df_initial['Quantity'] < 0][['CustomerID','Quantity',
                                                   'StockCode','Description','UnitPrice']]
for index, col in  df_check.iterrows():
    if df_initial[(df_initial['CustomerID'] == col[0]) & (df_initial['Quantity'] == -col[1]
                & (df_initial['Description'] == col[2])].shape[0] == 0:
        print(df_check.loc[index])
        print(15*'-'+'>'+' HYPOTHESIS NOT FULFILLED')
        break
```

```
CustomerID        14527
Quantity             -1
StockCode             D
Description    Discount
UnitPrice          27.5
Name: 141, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

In [14]:

```
df_check = df_initial[(df_initial['Quantity'] < 0) & (df_initial['Description'] != 'Discoun
                                ['CustomerID','Quantity','StockCode',
                                 'Description','UnitPrice']]

for index, col in  df_check.iterrows():
    if df_initial[(df_initial['CustomerID'] == col[0]) & (df_initial['Quantity'] == -col[1]
                & (df_initial['Description'] == col[2])].shape[0] == 0:
        print(index, df_check.loc[index])
        print(15*'-'+'>'+' HYPOTHESIS NOT FULFILLED')
        break
```

```
154 CustomerID                          15311
Quantity                               -1
StockCode                           35004C
Description    SET OF 3 COLOURED  FLYING DUCKS
UnitPrice                            4.65
Name: 154, dtype: object
---------------> HYPOTHESIS NOT FULFILLED
```

In [15]:

```
df_cleaned = df_initial.copy(deep = True)
df_cleaned['QuantityCanceled'] = 0

entry_to_remove = [] ; doubtfull_entry = []

for index, col in  df_initial.iterrows():
    if (col['Quantity'] > 0) or col['Description'] == 'Discount': continue
    df_test = df_initial[(df_initial['CustomerID'] == col['CustomerID']) &
                         (df_initial['StockCode']  == col['StockCode']) &
                         (df_initial['InvoiceDate'] < col['InvoiceDate']) &
                         (df_initial['Quantity']   > 0)].copy()

    # Cancelation WITHOUT counterpart
    if (df_test.shape[0] == 0):
        doubtfull_entry.append(index)

    # Cancelation WITH a counterpart
    elif (df_test.shape[0] == 1):
        index_order = df_test.index[0]
        df_cleaned.loc[index_order, 'QuantityCanceled'] = -col['Quantity']
        entry_to_remove.append(index)

    # Various counterparts exist in orders: we delete the last one
    elif (df_test.shape[0] > 1):
        df_test.sort_index(axis=0 ,ascending=False, inplace = True)
        for ind, val in df_test.iterrows():
            if val['Quantity'] < -col['Quantity']: continue
            df_cleaned.loc[ind, 'QuantityCanceled'] = -col['Quantity']
            entry_to_remove.append(index)
            break
```

In [16]:

```
print("entry_to_remove: {}".format(len(entry_to_remove)))
print("doubtfull_entry: {}".format(len(doubtfull_entry)))
```

```
entry_to_remove: 7521
doubtfull_entry: 1226
```

In [17]:

```
df_cleaned.drop(entry_to_remove, axis = 0, inplace = True)
df_cleaned.drop(doubtfull_entry, axis = 0, inplace = True)
remaining_entries = df_cleaned[(df_cleaned['Quantity'] < 0) & (df_cleaned['StockCode'] != '
print("nb of entries to delete: {}".format(remaining_entries.shape[0]))
remaining_entries[:5]
```

```
nb of entries to delete: 48
```

Out[17]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
77598	C542742	84535B	FAIRY CAKES NOTEBOOK A6 SIZE	-94	2011-01-31 16:26:00	0.65	15358	Ur King
90444	C544038	22784	LANTERN CREAM GAZEBO	-4	2011-02-15 11:32:00	4.95	14659	Ur King
111968	C545852	22464	HANGING METAL HEART LANTERN	-5	2011-03-07 13:49:00	1.65	14048	Ur King
116064	C546191	47566B	TEA TIME PARTY BUNTING	-35	2011-03-10 10:57:00	0.70	16422	Ur King
132642	C547675	22263	FELT EGG COSY LADYBIRD	-49	2011-03-24 14:07:00	0.66	17754	Ur King

In [18]:

```
df_cleaned[(df_cleaned['CustomerID'] == 14048) & (df_cleaned['StockCode'] == '22464')]
```

Out[18]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Q

In [19]:

```
list_special_codes = df_cleaned[df_cleaned['StockCode'].str.contains('^[a-zA-Z]+', regex=Tr
list_special_codes
```

Out[19]:

```
array(['POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT'],
      dtype=object)
```

In [20]:

```
for code in list_special_codes:
    print("{:<15} -> {:<30}".format(code, df_cleaned[df_cleaned['StockCode'] == code]['Desc
```

```
POST            -> POSTAGE
D               -> Discount
C2              -> CARRIAGE
M               -> Manual
BANK CHARGES    -> Bank Charges
PADS            -> PADS TO MATCH ALL CUSHIONS
DOT             -> DOTCOM POSTAGE
```

In [21]:

```
df_cleaned['TotalPrice'] = df_cleaned['UnitPrice'] * (df_cleaned['Quantity'] - df_cleaned['
df_cleaned.sort_values('CustomerID')[:5]
```

Out[21]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Cou
61619	541431	23166	MEDIUM CERAMIC TOP STORAGE JAR	74215	2011-01-18 10:01:00	1.04	12346	Un Kingc
148288	549222	22375	AIRLINE BAG VINTAGE JET SET BROWN	4	2011-04-07 10:43:00	4.25	12347	Icel
428971	573511	22698	PINK REGENCY TEACUP AND SAUCER	12	2011-10-31 12:25:00	2.95	12347	Icel
428970	573511	47559B	TEA TIME OVEN GLOVE	10	2011-10-31 12:25:00	1.25	12347	Icel
428969	573511	47567B	TEA TIME KITCHEN APRON	6	2011-10-31 12:25:00	5.95	12347	Icel

In [22]:

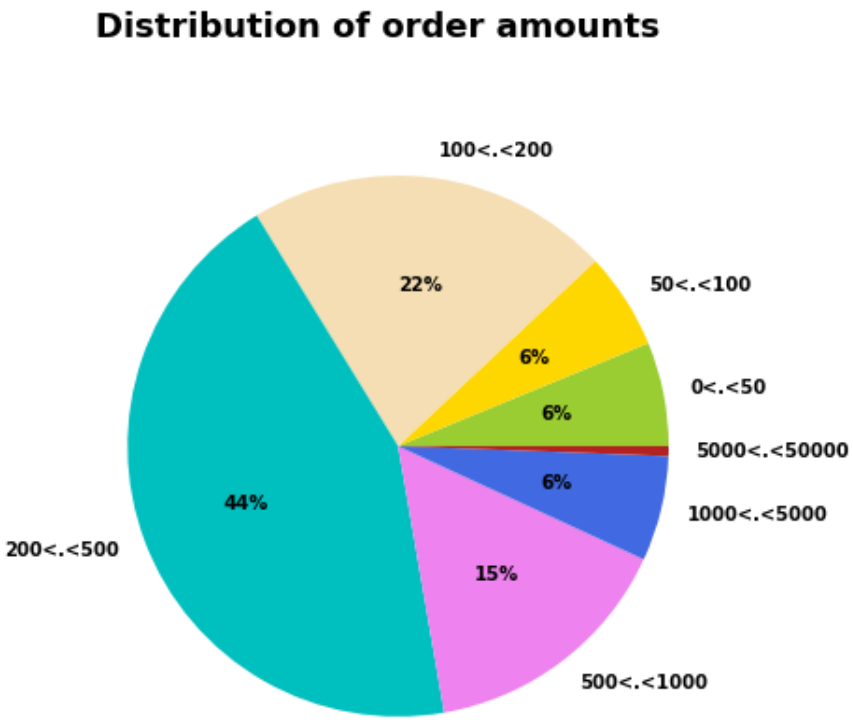
```
# sum of purchases / user & order
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['TotalPrice'].sum
basket_price = temp.rename(columns = {'TotalPrice':'Basket Price'})

# date of the order
df_cleaned['InvoiceDate_int'] = df_cleaned['InvoiceDate'].astype('int64')
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate_int'
df_cleaned.drop('InvoiceDate_int', axis = 1, inplace = True)
basket_price.loc[:, 'InvoiceDate'] = pd.to_datetime(temp['InvoiceDate_int'])

# selection of significant entries
basket_price = basket_price[basket_price['Basket Price'] > 0]
basket_price.sort_values('CustomerID')[:6]
```

Out[22]:

	CustomerID	InvoiceNo	Basket Price	InvoiceDate
1	12347	537626	711.79	2010-12-07 14:57:00.000001024
2	12347	542237	475.39	2011-01-26 14:29:59.999999744
3	12347	549222	636.25	2011-04-07 10:42:59.999999232
4	12347	556201	382.52	2011-06-09 13:01:00.000000256
5	12347	562032	584.91	2011-08-02 08:48:00.000000000
6	12347	573511	1294.32	2011-10-31 12:25:00.000001280

In [23]:

```
# Purchase count
price_range = [0, 50, 100, 200, 500, 1000, 5000, 50000]
count_price = []
for i, price in enumerate(price_range):
    if i == 0: continue
    val = basket_price[(basket_price['Basket Price'] < price) &
                       (basket_price['Basket Price'] > price_range[i-1])]['Basket Price'].c
    count_price.append(val)

# Representation of the number of purchases / amount
plt.rc('font', weight='bold')
f, ax = plt.subplots(figsize=(11, 6))
colors = ['yellowgreen', 'gold', 'wheat', 'c', 'violet', 'royalblue','firebrick']
labels = [ '{}<.<{}'.format(price_range[i-1], s) for i,s in enumerate(price_range) if i !=
sizes  = count_price
explode = [0.0 if sizes[i] < 100 else 0.0 for i in range(len(sizes))]
ax.pie(sizes, explode = explode, labels=labels, colors = colors,
       autopct = lambda x:'{:1.0f}%'.format(x) if x > 1 else '',
       shadow = False, startangle=0)
ax.axis('equal')
f.text(0.5, 1.01, "Distribution of order amounts", ha='center', fontsize = 18);
```



In [24]:

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

def keywords_inventory(dataframe, colonne = '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[colonne]:
        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("number of keywords in variable '{}': {}".format(colonne,len(category_keys)))
    return category_keys, keywords_roots, keywords_select, count_keywords
```

In [25]:

```
df_produits = pd.DataFrame(df_initial['Description'].unique()).rename(columns = {0:'Descrip
```

In [26]:

```
keywords, keywords_roots, keywords_select, count_keywords = keywords_inventory(df_produits)
```

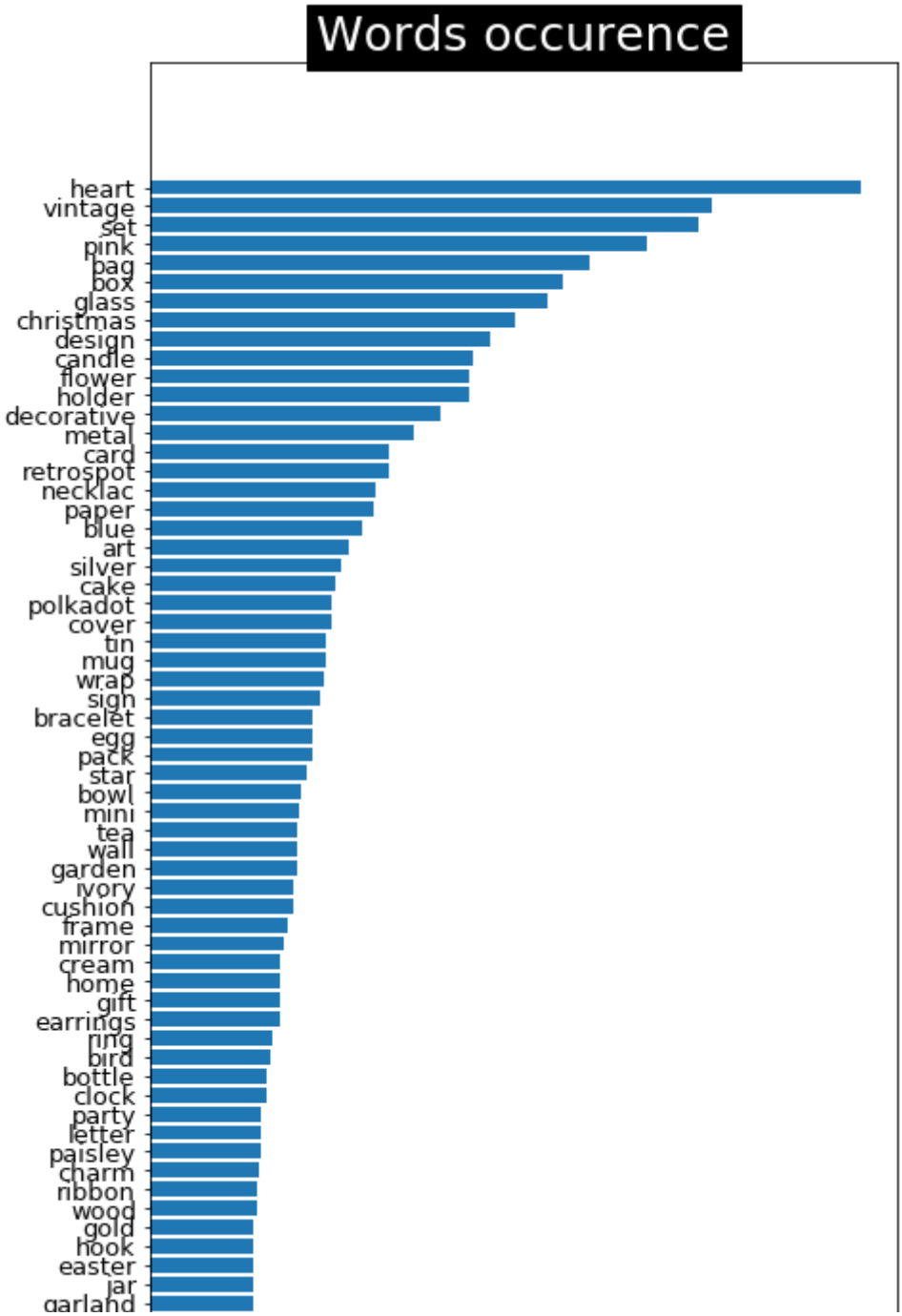
```
number of keywords in variable 'Description': 1482
```

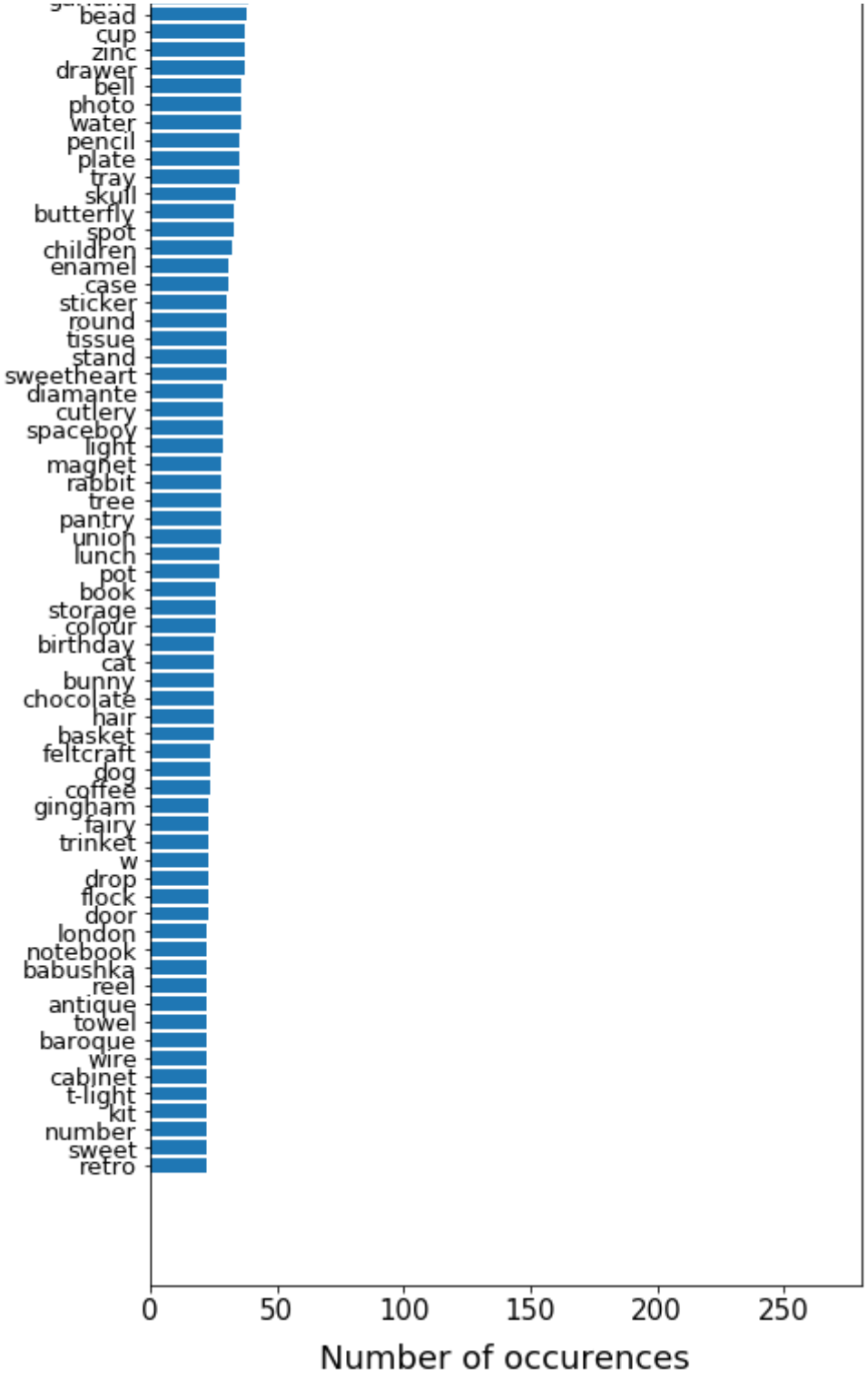
In [27]:

```
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)
```

In [28]:

```
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("Number 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()
```





In [29]:

```
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('Preserved words:', len(list_products))
```

```
Preserved words: 193
```

In [30]:

```
liste_produits = df_cleaned['Description'].unique()
#print(liste_produits[0:2])
X = pd.DataFrame()
for key, occurence in list_products:
    X.loc[:, key] = list(map(lambda x:int(key.upper() in x), liste_produits))
#print(X[0:1])
```

In [31]:

```
threshold = [0, 1, 2, 3, 5, 10]
label_col = []
for i in range(len(threshold)):
    if i == len(threshold)-1:
        col = '.>{}'.format(threshold[i])
    else:
        col = '{}<.<{}'.format(threshold[i],threshold[i+1])
    #print(i)
    #print(col)
    label_col.append(col)
    X.loc[:, col] = 0

for i, prod in enumerate(liste_produits):
    prix = df_cleaned[ df_cleaned['Description'] == prod]['UnitPrice'].mean()
    #print (prix)
    j = 0
    while prix > threshold[j]:
        j+=1
        if j == len(threshold): break
    X.loc[i, label_col[j-1]] = 1
```

In [32]:

```
print("{:<8} {:<20} \n".format('range', 'number of products') + 20*'-')
for i in range(len(threshold)):
    if i == len(threshold)-1:
        col = '.>{}'.format(threshold[i])
    else:
        col = '{}<.<{}'.format(threshold[i],threshold[i+1])
    print("{:<10}  {:<20}".format(col, X.loc[:, col].sum()))
```

```
range    number of products
--------------------
0<.<1       964
1<.<2       1009
2<.<3       673
3<.<5       606
5<.<10      470
.>10        156
```

In [33]:

```
matrix = X.as_matrix()
for n_clusters in range(3,10):
    kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init=30)
    kmeans.fit(matrix)
    clusters = kmeans.predict(matrix)
    silhouette_avg = silhouette_score(matrix, clusters)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_a
```

```
For n_clusters = 3 The average silhouette_score is : 0.09751688498995637
For n_clusters = 4 The average silhouette_score is : 0.12609893747265383
For n_clusters = 5 The average silhouette_score is : 0.1466257603527048
For n_clusters = 6 The average silhouette_score is : 0.14861342533737415
For n_clusters = 7 The average silhouette_score is : 0.15083317916457992
For n_clusters = 8 The average silhouette_score is : 0.15054954899418788
For n_clusters = 9 The average silhouette_score is : 0.1480858139817441
```

In [34]:

```
n_clusters = 5
silhouette_avg = -1
while silhouette_avg < 0.145:
    kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init=30)
    kmeans.fit(matrix)
    clusters = kmeans.predict(matrix)
    silhouette_avg = silhouette_score(matrix, clusters)

    #km = kmodes.KModes(n_clusters = n_clusters, init='Huang', n_init=2, verbose=0)
    #clusters = km.fit_predict(matrix)
    #silhouette_avg = silhouette_score(matrix, clusters)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_a
```

```
For n_clusters = 5 The average silhouette_score is : 0.1452148389646187
```

In [35]:

```
pd.Series(clusters).value_counts()
```

Out[35]:

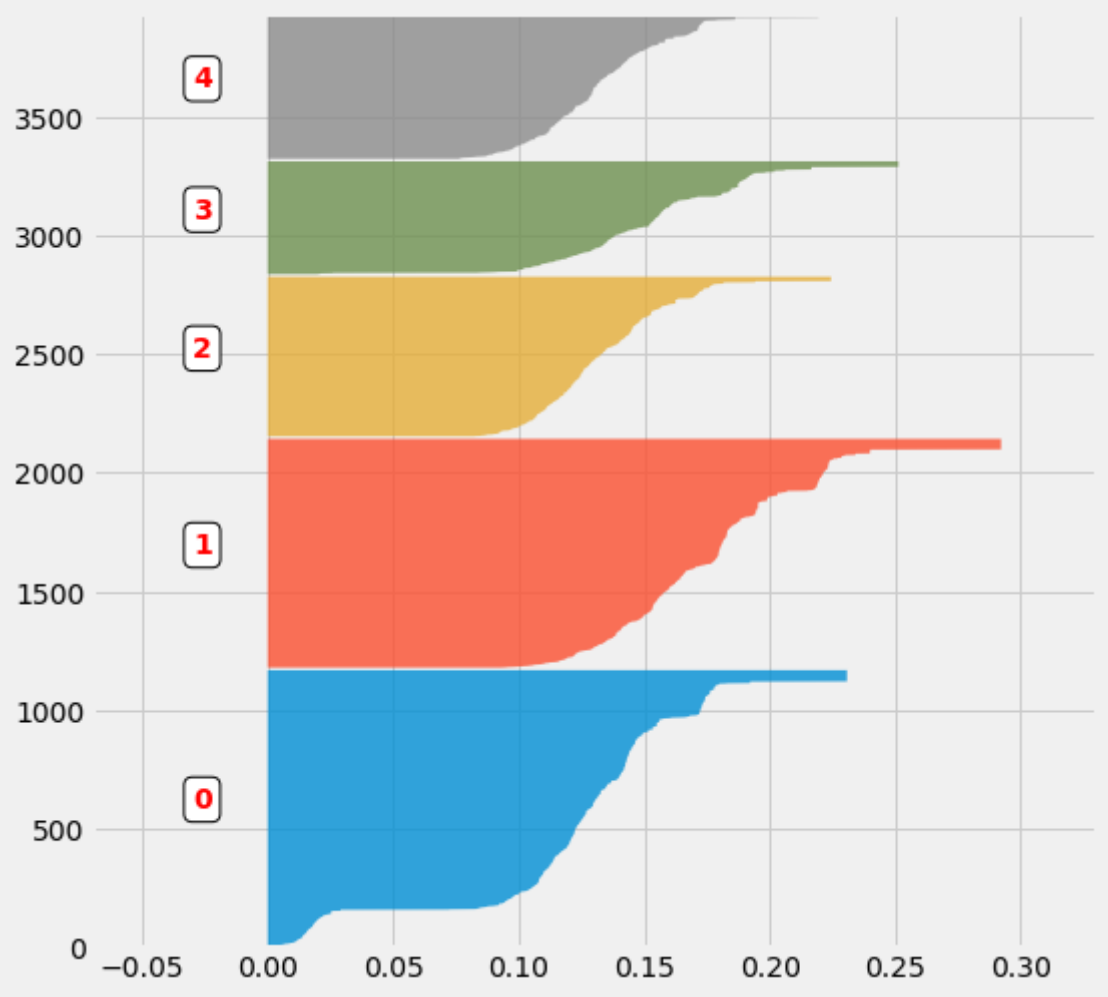
```
0    1159
1     964
2     673
4     606
3     476
dtype: int64
```

In [36]:

```
def graph_component_silhouette(n_clusters, lim_x, mat_size, sample_silhouette_values, clust
    #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
        #color = cm.spectral(float(i) / n_clusters) facecolor=color, edgecolor=color,
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values, al

        # 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 =
                bbox=dict(facecolor='white', edgecolor='black', boxstyle='round, pad=0.3'))

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

In [37]:

```
# define individual silouhette scores
sample_silhouette_values = silhouette_samples(matrix, clusters)

# and do the graph
graph_component_silhouette(n_clusters, [-0.07, 0.33], len(X), sample_silhouette_values, clu
```



In [38]:

```
liste = pd.DataFrame(liste_produits)
liste_words = [word for (word, occurence) in list_products]

occurence = [dict() for _ in range(n_clusters)]

for i in range(n_clusters):
    liste_cluster = liste.loc[clusters == i]
    for word in liste_words:
        if word in ['art', 'set', 'heart', 'pink', 'blue', 'tag']: continue
        occurence[i][word] = sum(liste_cluster.loc[:, 0].str.contains(word.upper()))
```

In [39]:

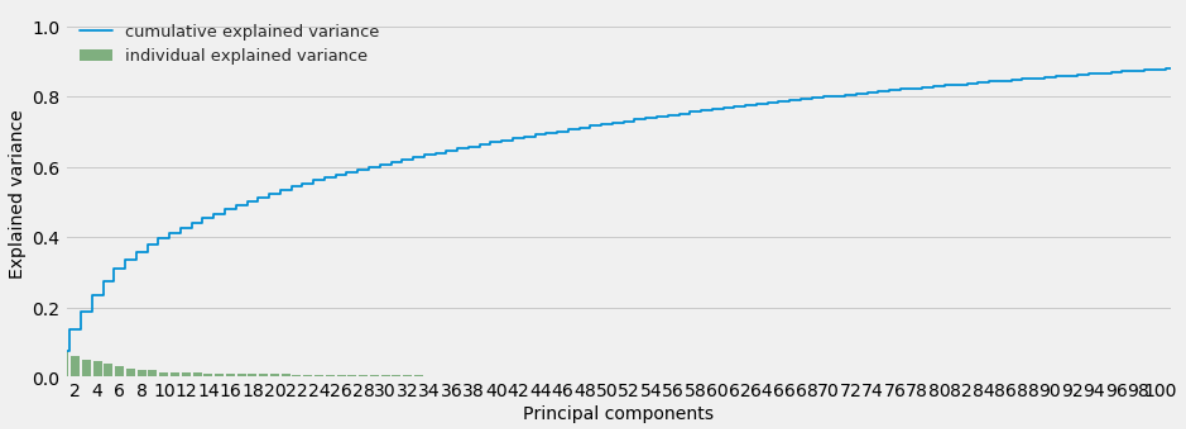
```
def random_color_func(word=None, font_size=None, position=None,
                      orientation=None, font_path=None, random_state=None):
    h = int(360.0 * tone / 255.0)
    s = int(100.0 * 255.0 / 255.0)
    l = int(100.0 * float(random_state.randint(70, 120)) / 255.0)
    return "hsl({}, {}%, {}%)".format(h, s, l)

def make_wordcloud(liste, increment):
    ax1 = fig.add_subplot(4,2,increment)
    words = dict()
    trunc_occurences = liste[0:150]
    for s in trunc_occurences:
        words[s[0]] = s[1]

    wordcloud = WordCloud(width=1000,height=400, background_color='lightgrey',
                          max_words=1628,relative_scaling=1,
                          color_func = random_color_func,
                          normalize_plurals=False)
    wordcloud.generate_from_frequencies(words)
    ax1.imshow(wordcloud, interpolation="bilinear")
    ax1.axis('off')
    plt.title('cluster n{}'.format(increment-1))

fig = plt.figure(1, figsize=(14,14))
color = [0, 160, 130, 95, 280, 40, 330, 110, 25]
for i in range(n_clusters):
    list_cluster_occurences = occurence[i]

    tone = color[i] # define the color of the words
    liste = []
    for key, value in list_cluster_occurences.items():
        liste.append([key, value])
    liste.sort(key = lambda x:x[1], reverse = True)
    make_wordcloud(liste, i+1)
```

In [40]:

```
pca = PCA()
pca.fit(matrix)
pca_samples = pca.transform(matrix)
```

In [41]:

```
fig, ax = plt.subplots(figsize=(14, 5))
sns.set(font_scale=1)
plt.step(range(matrix.shape[1]), pca.explained_variance_ratio_.cumsum(), where='mid',
         label='cumulative explained variance')
sns.barplot(np.arange(1,matrix.shape[1]+1), pca.explained_variance_ratio_, alpha=0.5, color
            label='individual explained variance')
plt.xlim(0, 100)

ax.set_xticklabels([s if int(s.get_text())%2 == 0 else '' for s in ax.get_xticklabels()])

plt.ylabel('Explained variance', fontsize = 14)
plt.xlabel('Principal components', fontsize = 14)
plt.legend(loc='upper left', fontsize = 13);
```

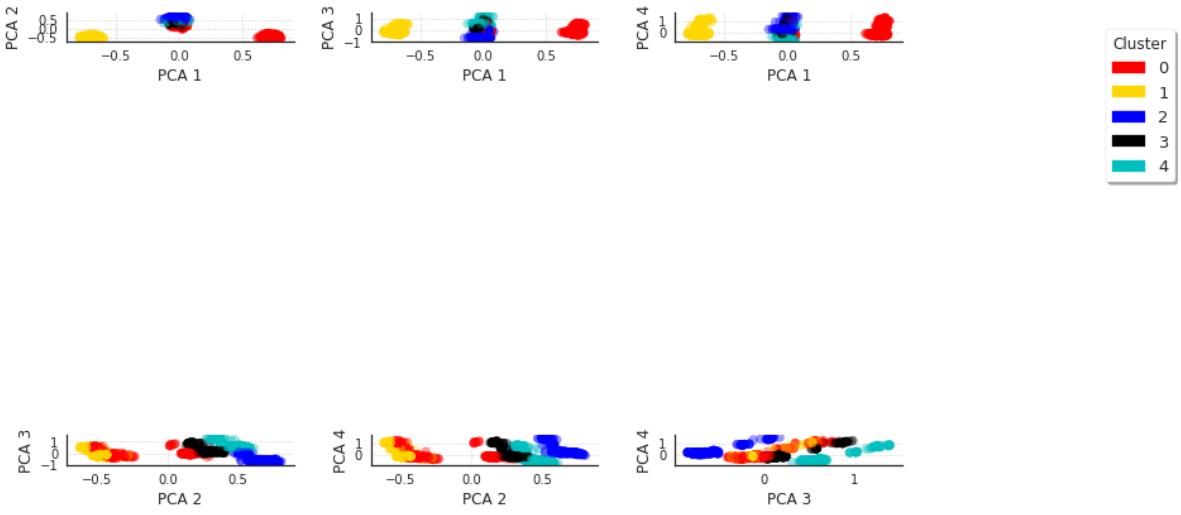


In [42]:

```
pca = PCA(n_components=50)
matrix_9D = pca.fit_transform(matrix)
mat = pd.DataFrame(matrix_9D)
mat['cluster'] = pd.Series(clusters)
```

In [43]:

```
import matplotlib.patches as mpatches

sns.set_style("white")
sns.set_context("notebook", font_scale=1, rc={"lines.linewidth": 2.5})

LABEL_COLOR_MAP = {0:'r', 1:'gold', 2:'b', 3:'k', 4:'c', 5:'g'}
label_color = [LABEL_COLOR_MAP[l] for l in mat['cluster']]

fig = plt.figure(figsize = (12,10))
increment = 0
for ix in range(4):
    for iy in range(ix+1, 4):
        increment += 1
        ax = fig.add_subplot(3,3,increment)
        ax.scatter(mat[ix], mat[iy], c= label_color, alpha=0.4)
        plt.ylabel('PCA {}'.format(iy+1), fontsize = 12)
        plt.xlabel('PCA {}'.format(ix+1), fontsize = 12)
        ax.yaxis.grid(color='lightgray', linestyle=':')
        ax.xaxis.grid(color='lightgray', linestyle=':')
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)

        if increment == 9: break
    if increment == 9: break


comp_handler = []
for i in range(5):
    comp_handler.append(mpatches.Patch(color = LABEL_COLOR_MAP[i], label = i))

plt.legend(handles=comp_handler, bbox_to_anchor=(1.1, 0.97),
           title='Cluster',
           shadow = True, frameon = True, framealpha = 1,fontsize = 13,
           bbox_transform = plt.gcf().transFigure) #facecolor = 'lightgrey',

plt.tight_layout()
```



In [44]:

```
corresp = dict()
for key, val in zip (liste_produits, clusters):
    corresp[key] = val

df_cleaned['categ_product'] = df_cleaned.loc[:, 'Description'].map(corresp)
df_cleaned[['InvoiceNo', 'Description',
            'categ_product']][:10]
```

Out[44]:

	InvoiceNo	Description	categ_product
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	2
1	536365	WHITE METAL LANTERN	4
2	536365	CREAM CUPID HEARTS COAT HANGER	4
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	4
4	536365	RED WOOLLY HOTTIE WHITE HEART.	4
5	536365	SET 7 BABUSHKA NESTING BOXES	3
6	536365	GLASS STAR FROSTED T-LIGHT HOLDER	4
7	536366	HAND WARMER UNION JACK	2
8	536366	HAND WARMER RED POLKA DOT	0
9	536367	ASSORTED COLOUR BIRD ORNAMENT	0

In [45]:

```
for i in range(5):
    col = 'categ_{}'.format(i)
    df_temp = df_cleaned[df_cleaned['categ_product'] == i]
    price_temp = df_temp['UnitPrice'] * (df_temp['Quantity'] - df_temp['QuantityCanceled'])
    price_temp = price_temp.apply(lambda x:x if x > 0 else 0)
    df_cleaned.loc[:, col] = price_temp
    df_cleaned[col].fillna(0, inplace = True)


df_cleaned[['InvoiceNo', 'Description',
            'categ_product', 'categ_0', 'categ_1', 'categ_2', 'categ_3','categ_4']][:10]
```

Out[45]:

	InvoiceNo	Description	categ_product	categ_0	categ_1	categ_2	categ_3	categ_4
0	536365	WHITE HANGING HEART T-LIGHT HOLDER	2	0.00	0.0	15.3	0.0	0.00
1	536365	WHITE METAL LANTERN	4	0.00	0.0	0.0	0.0	20.34
2	536365	CREAM CUPID HEARTS COAT HANGER	4	0.00	0.0	0.0	0.0	22.00
3	536365	KNITTED UNION FLAG HOT WATER BOTTLE	4	0.00	0.0	0.0	0.0	20.34
4	536365	RED WOOLLY HOTTIE WHITE HEART.	4	0.00	0.0	0.0	0.0	20.34
5	536365	SET 7 BABUSHKA NESTING BOXES	3	0.00	0.0	0.0	15.3	0.00
6	536365	GLASS STAR FROSTED T-LIGHT HOLDER	4	0.00	0.0	0.0	0.0	25.50
7	536366	HAND WARMER UNION JACK	2	0.00	0.0	11.1	0.0	0.00
8	536366	HAND WARMER RED POLKA DOT	0	11.10	0.0	0.0	0.0	0.00
9	536367	ASSORTED COLOUR BIRD ORNAMENT	0	54.08	0.0	0.0	0.0	0.00

In [46]:

```
# sum of purchases / user & order
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['TotalPrice'].sum
basket_price = temp.rename(columns = {'TotalPrice':'Basket Price'})

# percentage of the price of the order / product category
for i in range(5):
    col = 'categ_{}'.format(i)
    temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)[col].sum()
    basket_price.loc[:, col] = temp

# date of the order

df_cleaned['InvoiceDate_int'] = df_cleaned['InvoiceDate'].astype('int64')
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate_int'
df_cleaned.drop('InvoiceDate_int', axis = 1, inplace = True)
basket_price.loc[:, 'InvoiceDate'] = pd.to_datetime(temp['InvoiceDate_int'])

# selection of significant entries:
basket_price = basket_price[basket_price['Basket Price'] > 0]
basket_price.sort_values('CustomerID', ascending = True)[:5]
```

Out[46]:

	CustomerID	InvoiceNo	Basket Price	categ_0	categ_1	categ_2	categ_3	categ_4	Invoice
1	12347	537626	711.79	187.20	23.40	83.40	124.44	293.35	2010-1 14:57:00.00000
2	12347	542237	475.39	168.75	84.34	53.10	0.00	169.20	2011-0 14:29:59.99999
3	12347	549222	636.25	369.15	81.00	71.10	0.00	115.00	2011-0 10:42:59.99999
4	12347	556201	382.52	74.40	41.40	78.06	19.90	168.76	2011-0 13:01:00.00000
5	12347	562032	584.91	147.95	61.30	119.70	97.80	158.16	2011-0 08:48:00.00000

In [47]:

```
print(basket_price['InvoiceDate'].min(), '->',  basket_price['InvoiceDate'].max())
```

```
2010-12-01 08:26:00 -> 2011-12-09 12:50:00
```

In [48]:

```
set_entrainement = basket_price[basket_price['InvoiceDate'] < datetime.date(2011,10,1)]
set_test         = basket_price[basket_price['InvoiceDate'] >= datetime.date(2011,10,1)]
basket_price = set_entrainement.copy(deep = True)
```

In [49]:

```
# of visits and stats on cart amount / users
transactions_per_user=basket_price.groupby(by=['CustomerID'])['Basket Price'].agg(['count',
                                                                                    'max','m
for i in range(5):
    col = 'categ_{}'.format(i)
    transactions_per_user.loc[:,col] = basket_price.groupby(by=['CustomerID'])[col].sum() /
                                            transactions_per_user['sum']*100

transactions_per_user.reset_index(drop = False, inplace = True)
basket_price.groupby(by=['CustomerID'])['categ_0'].sum()
transactions_per_user.sort_values('CustomerID', ascending = True)[:5]
```

Out[49]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	
0	12347	5	382.52	711.79	558.172000	2790.86	33.948317	10.442659	14.524555	8
1	12348	4	227.44	892.80	449.310000	1797.24	61.983931	38.016069	0.000000	(
2	12350	1	334.40	334.40	334.400000	334.40	60.406699	11.692584	27.900718	(
3	12352	6	144.35	840.30	345.663333	2073.98	66.125517	0.491808	3.370331	14
4	12353	1	89.00	89.00	89.000000	89.00	57.752809	0.000000	19.887640	22

In [50]:

```
last_date = basket_price['InvoiceDate'].max().date()

first_registration = pd.DataFrame(basket_price.groupby(by=['CustomerID'])['InvoiceDate'].mi
last_purchase      = pd.DataFrame(basket_price.groupby(by=['CustomerID'])['InvoiceDate'].ma

test  = first_registration.applymap(lambda x:(last_date - x.date()).days)
test2 = last_purchase.applymap(lambda x:(last_date - x.date()).days)

transactions_per_user.loc[:, 'LastPurchase'] = test2.reset_index(drop = False)['InvoiceDate
transactions_per_user.loc[:, 'FirstPurchase'] = test.reset_index(drop = False)['InvoiceDate

transactions_per_user[:5]
```

Out[50]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	
0	12347	5	382.52	711.79	558.172000	2790.86	33.948317	10.442659	14.524555	8
1	12348	4	227.44	892.80	449.310000	1797.24	61.983931	38.016069	0.000000	(
2	12350	1	334.40	334.40	334.400000	334.40	60.406699	11.692584	27.900718	(
3	12352	6	144.35	840.30	345.663333	2073.98	66.125517	0.491808	3.370331	14
4	12353	1	89.00	89.00	89.000000	89.00	57.752809	0.000000	19.887640	22

In [51]:

```
n1 = transactions_per_user[transactions_per_user['count'] == 1].shape[0]
n2 = transactions_per_user.shape[0]
print("No. customers with single purchase: {:<2}/{:<5} ({:<2.2f}%)".format(n1,n2,n1/n2*100)
```

```
No. customers with single purchase: 1445/3608  (40.05%)
```

In [52]:

```
list_cols = ['count','min','max','mean','categ_0','categ_1','categ_2','categ_3','categ_4']
#_____________________________________________________________
selected_customers = transactions_per_user.copy(deep = True)
matrix = selected_customers[list_cols].as_matrix()
```

In [53]:

```
scaler = StandardScaler()
scaler.fit(matrix)
print('variables mean values: \n' + 90*'-' + '\n' , scaler.mean_)
scaled_matrix = scaler.transform(matrix)
```

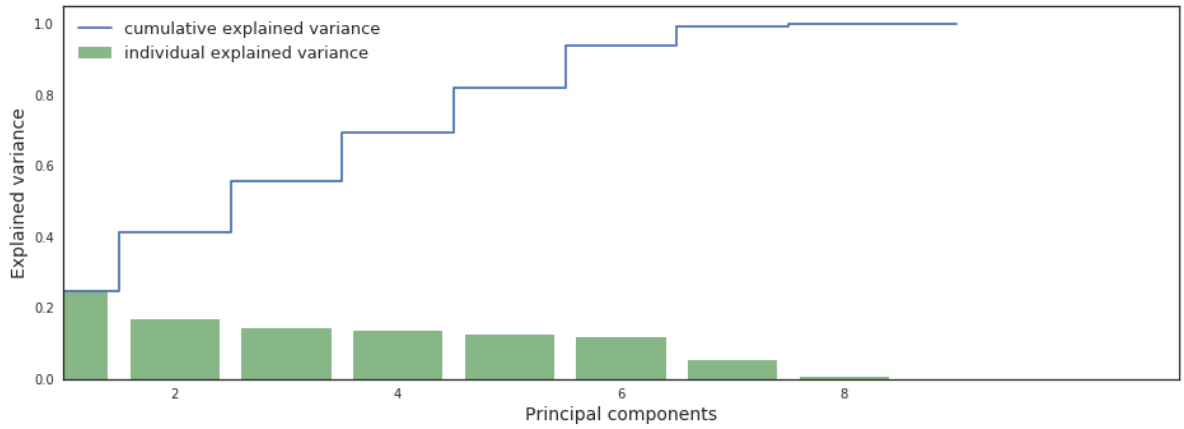
```
variables mean values:
------------------------------------------------------------------------------
--------------
 [  3.62305987 259.93189634 556.26687999 377.06036244  32.75310053
  13.98907929  21.19884856  15.6945421   16.37327913]
```

In [54]:

```
pca = PCA()
pca.fit(scaled_matrix)
pca_samples = pca.transform(scaled_matrix)
```

In [55]:

```
fig, ax = plt.subplots(figsize=(14, 5))
sns.set(font_scale=1)
plt.step(range(matrix.shape[1]), pca.explained_variance_ratio_.cumsum(), where='mid',
         label='cumulative explained variance')
sns.barplot(np.arange(1,matrix.shape[1]+1), pca.explained_variance_ratio_, alpha=0.5, color
            label='individual explained variance')
plt.xlim(0, 10)

ax.set_xticklabels([s if int(s.get_text())%2 == 0 else '' for s in ax.get_xticklabels()])

plt.ylabel('Explained variance', fontsize = 14)
plt.xlabel('Principal components', fontsize = 14)
plt.legend(loc='best', fontsize = 13);
```



In [56]:

```
n_clusters = 11
kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init=100)
kmeans.fit(scaled_matrix)
clusters_clients = kmeans.predict(scaled_matrix)
silhouette_avg = silhouette_score(scaled_matrix, clusters_clients)
print('silhouette score: {:<.3f}'.format(silhouette_avg))
```

```
silhouette score: 0.218
```

In [57]:

```
pd.DataFrame(pd.Series(clusters_clients).value_counts(), columns = ['number of clients']).T
```

Out[57]:

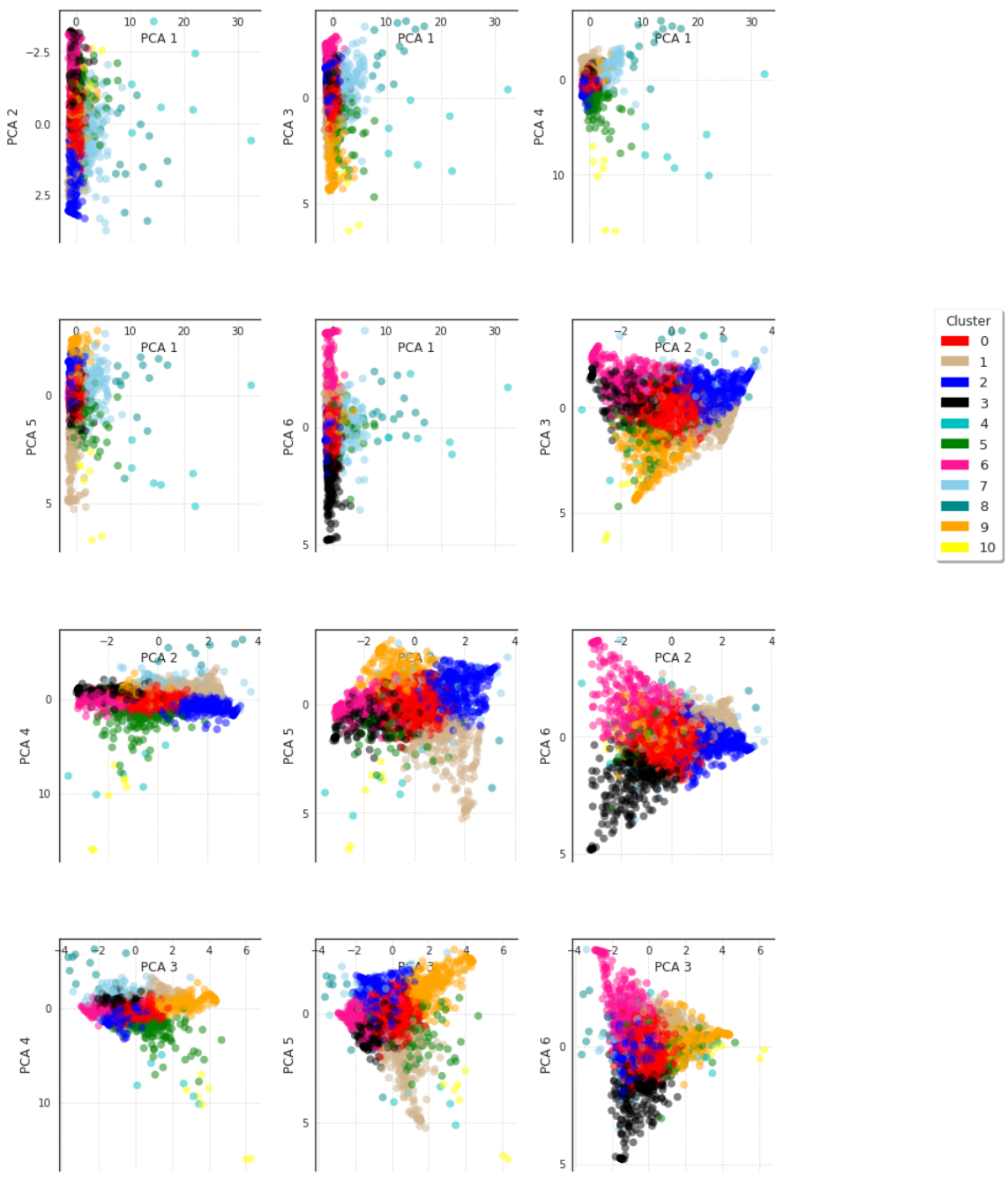
	0	2	6	9	3	1	7	5	8	10	4
number of clients	1553	502	334	294	291	265	191	151	13	7	7

In [58]:

```
pca = PCA(n_components=6)
matrix_3D = pca.fit_transform(scaled_matrix)
mat = pd.DataFrame(matrix_3D)
mat['cluster'] = pd.Series(clusters_clients)
```

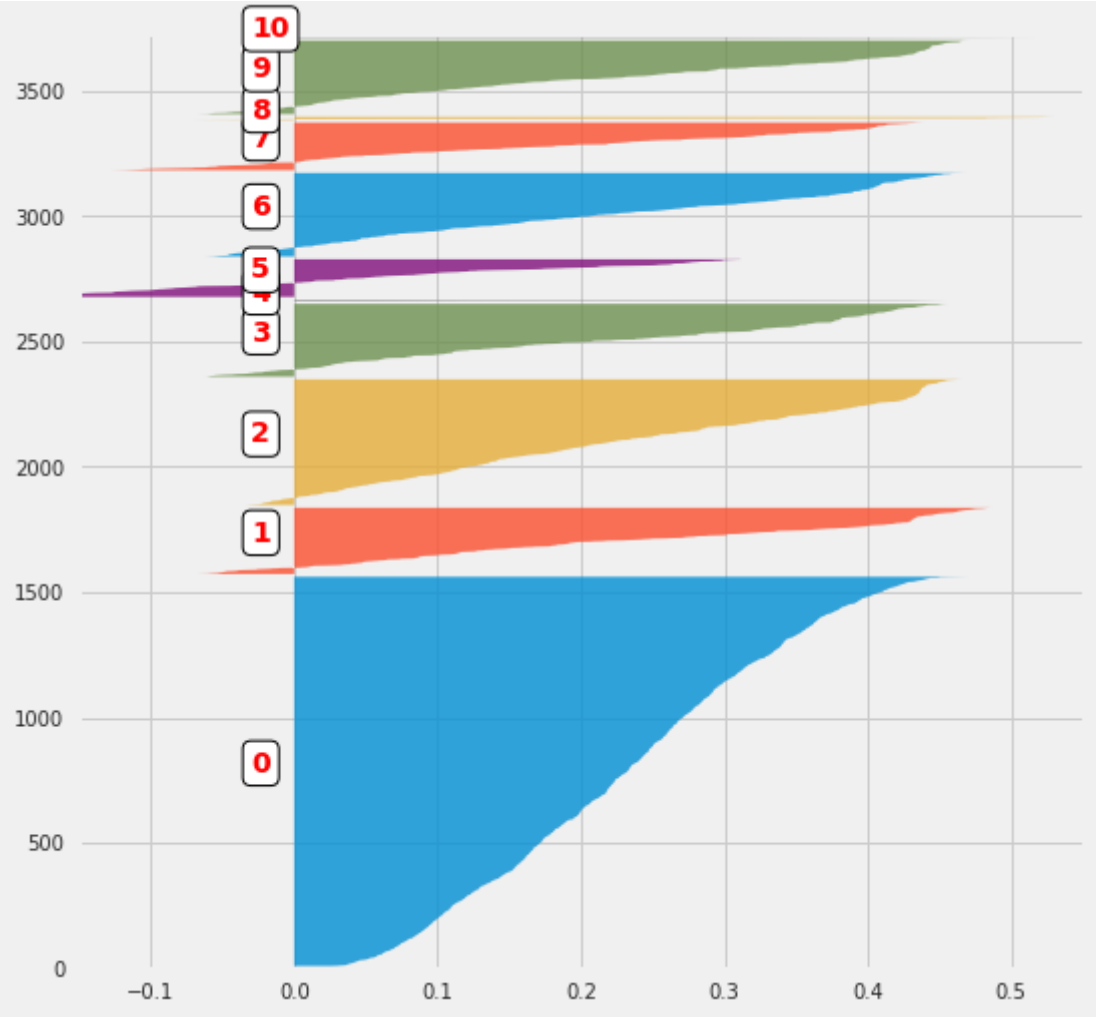
In [59]:

```
import matplotlib.patches as mpatches

sns.set_style("white")
sns.set_context("notebook", font_scale=1, rc={"lines.linewidth": 2.5})

LABEL_COLOR_MAP = {0:'r', 1:'tan', 2:'b', 3:'k', 4:'c', 5:'g', 6:'deeppink', 7:'skyblue', 8
                   9:'orange',
                   10:'yellow', 11:'tomato', 12:'seagreen'}
label_color = [LABEL_COLOR_MAP[l] for l in mat['cluster']]

fig = plt.figure(figsize = (12,10))
increment = 0
for ix in range(6):
    for iy in range(ix+1, 6):
        increment += 1
        ax = fig.add_subplot(4,3,increment)
        ax.scatter(mat[ix], mat[iy], c= label_color, alpha=0.5)
        plt.ylabel('PCA {}'.format(iy+1), fontsize = 12)
        plt.xlabel('PCA {}'.format(ix+1), fontsize = 12)
        ax.yaxis.grid(color='lightgray', linestyle=':')
        ax.xaxis.grid(color='lightgray', linestyle=':')
        ax.spines['right'].set_visible(False)
        ax.spines['top'].set_visible(False)

        if increment == 12: break
    if increment == 12: break

#_____________________________________________
# I set the legend: abreviation -> airline name
comp_handler = []
for i in range(n_clusters):
    comp_handler.append(mpatches.Patch(color = LABEL_COLOR_MAP[i], label = i))

plt.legend(handles=comp_handler, bbox_to_anchor=(1.1, 0.9),
           title='Cluster',
           shadow = True, frameon = True, framealpha = 1,
           fontsize = 13, bbox_transform = plt.gcf().transFigure) #facecolor = 'lightgrey',

plt.tight_layout()
```


PCA 1
PCA 2
PCA 3
PCA 4
PCA 5
PCA 6
Cluster
0
1
2
3
4
5
6
7
8
9
10

In [60]:

```
sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
#_____________________________________
# define individual silhouette scores
sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
#___________________
# and do the graph
graph_component_silhouette(n_clusters, [-0.15, 0.55], len(scaled_matrix), sample_silhouette
                           clusters_clients)
```



In [61]:

```
selected_customers.loc[:, 'cluster'] = clusters_clients
```

In [62]:

```
merged_df = pd.DataFrame()
for i in range(n_clusters):
    test = pd.DataFrame(selected_customers[selected_customers['cluster'] == i].mean())
    test = test.T.set_index('cluster', drop = True)
    test['size'] = selected_customers[selected_customers['cluster'] == i].shape[0]
    merged_df = pd.concat([merged_df, test])
#_____________________________________________________
merged_df.drop('CustomerID', axis = 1, inplace = True)
print('number of customers:', merged_df['size'].sum())

merged_df = merged_df.sort_values('sum')
```

```
number of customers: 3608
```

In [63]:

```
liste_index = []
for i in range(5):
    column = 'categ_{}'.format(i)
    liste_index.append(merged_df[merged_df[column] > 45].index.values[0])

liste_index_reordered = liste_index
liste_index_reordered += [ s for s in merged_df.index if s not in liste_index]

merged_df = merged_df.reindex(index = liste_index_reordered)
merged_df = merged_df.reset_index(drop = False)
display(merged_df[['cluster', 'count', 'min', 'max', 'mean', 'sum', 'categ_0',
                   'categ_1', 'categ_2', 'categ_3', 'categ_4', 'size']])
```

Out[ ]:

	cluster	count	min	max	mean	sum	categ_0	ca
0	2.0	2.432271	198.114363	326.923946	255.382523	661.292114	66.451745	8.87
1	1.0	2.260377	193.966755	316.250415	247.336663	593.788566	22.081663	54.8
2	9.0	2.591837	209.375646	382.809660	292.227928	823.159728	17.664849	6.97
3	6.0	2.482036	191.252725	307.080689	243.361162	620.928204	17.160023	5.2
4	3.0	2.144330	202.483505	339.846014	264.699716	665.022405	17.008286	6.6
5	0.0	3.259498	223.028050	457.820812	331.429539	1085.060413	32.743319	13.6
6	7.0	1.712042	1033.485759	1395.821209	1197.460012	2175.600634	35.608706	12.2
7	8.0	1.692308	3253.388462	4380.010000	3794.797051	6250.506154	29.560482	21.7
8	5.0	18.503311	85.567682	1699.433576	585.954054	10333.611457	30.517202	12.1
9	10.0	92.000000	10.985714	1858.250000	374.601553	34845.105714	33.256402	13.1
10	4.0	26.857143	510.302857	20131.802857	5514.816882	113654.117143	30.232736	7.87

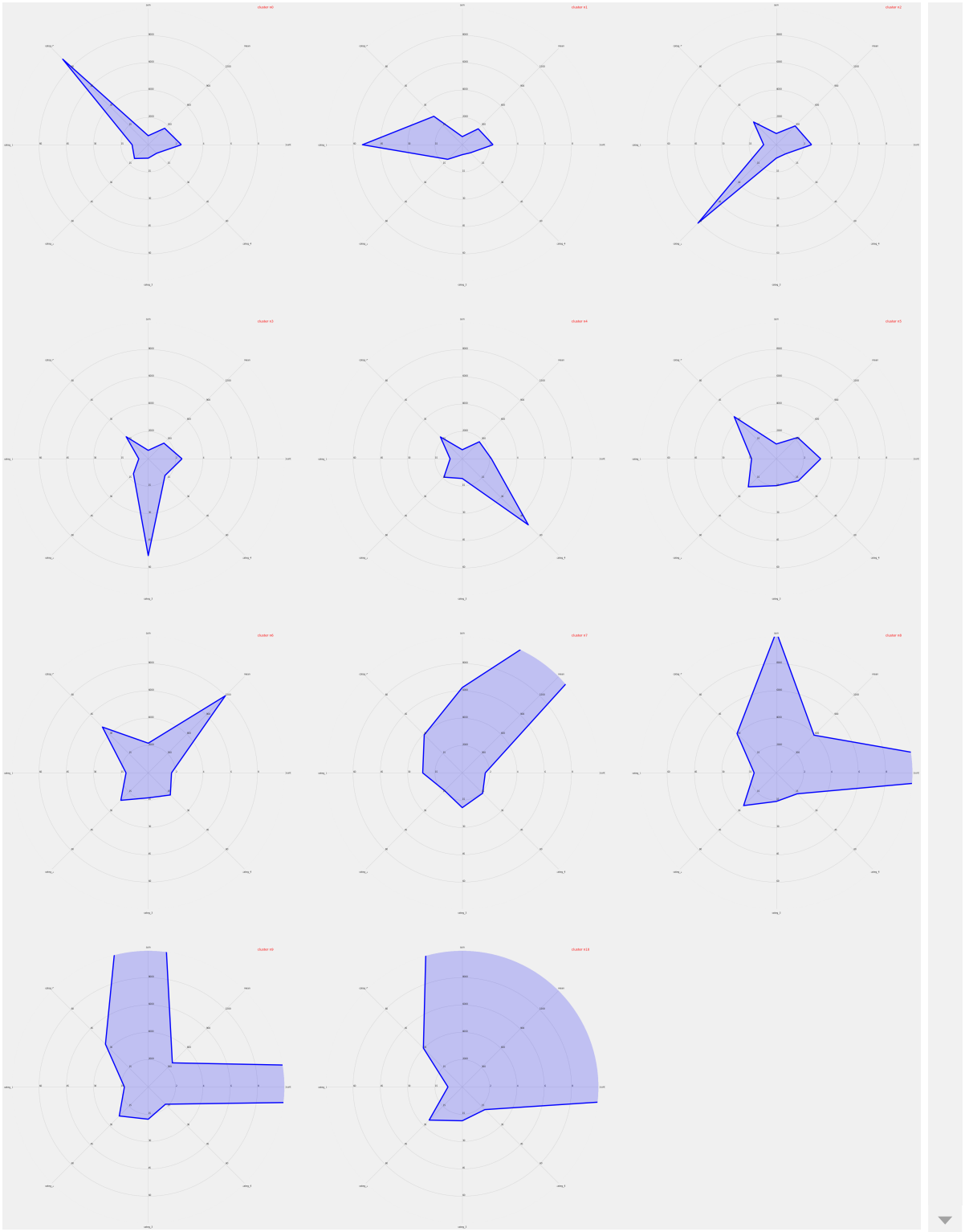
In [64]:

```
def _scale_data(data, ranges):
    (x1, x2) = ranges[0]
    d = data[0]
    return [(d - y1) / (y2 - y1) * (x2 - x1) + x1 for d, (y1, y2) in zip(data, ranges)]

class RadarChart():
    def __init__(self, fig, location, sizes, variables, ranges, n_ordinate_levels = 6):

        angles = np.arange(0, 360, 360./len(variables))

        ix, iy = location[:] ; size_x, size_y = sizes[:]

        axes = [fig.add_axes([ix, iy, size_x, size_y], polar = True,
        label = "axes{}".format(i)) for i in range(len(variables))]

        _, text = axes[0].set_thetagrids(angles, labels = variables)

        for txt, angle in zip(text, angles):
            if angle > -1 and angle < 181:
                txt.set_rotation(angle - 90)
            else:
                txt.set_rotation(angle - 270)

        for ax in axes[1:]:
            ax.patch.set_visible(False)
            ax.xaxis.set_visible(False)
            ax.grid("off")

        for i, ax in enumerate(axes):
            grid = np.linspace(*ranges[i],num = n_ordinate_levels)
            grid_label = [""]+["{:.0f}".format(x) for x in grid[1:-1]]
            ax.set_rgrids(grid, labels = grid_label, angle = angles[i])
            ax.set_ylim(*ranges[i])

        self.angle = np.deg2rad(np.r_[angles, angles[0]])
        self.ranges = ranges
        self.ax = axes[0]

    def plot(self, data, *args, **kw):
        sdata = _scale_data(data, self.ranges)
        self.ax.plot(self.angle, np.r_[sdata, sdata[0]], *args, **kw)

    def fill(self, data, *args, **kw):
        sdata = _scale_data(data, self.ranges)
        self.ax.fill(self.angle, np.r_[sdata, sdata[0]], *args, **kw)

    def legend(self, *args, **kw):
        self.ax.legend(*args, **kw)

    def title(self, title, *args, **kw):
        self.ax.text(0.9, 1, title, transform = self.ax.transAxes, *args, **kw)
```

In [65]:

```
fig = plt.figure(figsize=(50,50))

attributes = ['count', 'mean', 'sum', 'categ_0', 'categ_1', 'categ_2', 'categ_3', 'categ_4'
ranges = [[0.01, 10], [0.01, 1500], [0.01, 10000], [0.01, 75], [0.01, 75], [0.01, 75], [0.0
index  = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

n_groups = n_clusters ; i_cols = 3
i_rows = n_groups//i_cols
size_x, size_y = (1/i_cols), (1/i_rows)

for ind in range(n_clusters):
    ix = ind%3 ; iy = i_rows - ind//3
    pos_x = ix*(size_x + 0.05) ; pos_y = iy*(size_y + 0.05)
    location = [pos_x, pos_y]  ; sizes = [size_x, size_y]
    #_____________________________________________________
    data = np.array(merged_df.loc[index[ind], attributes])
    #print (data)
    radar = RadarChart(fig, location, sizes, attributes, ranges)
    radar.plot(data, color = 'b', linewidth=5.0)
    radar.fill(data, alpha = 0.2, color = 'b')
    radar.title(title = 'cluster n{}'.format(index[ind]), color = 'r')
    ind += 1
```

```
/usr/local/lib/python3.5/dist-packages/matplotlib/cbook/deprecation.py:107:
MatplotlibDeprecationWarning: Passing one of 'on', 'true', 'off', 'false' as
a boolean is deprecated; use an actual boolean (True/False) instead.
  warnings.warn(message, mplDeprecation, stacklevel=1)
```



In [66]:

```
class Class_Fit(object):
    def __init__(self, clf, params=None):
        if params:
            self.clf = clf(**params)
        else:
            self.clf = clf()

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def grid_search(self, parameters, Kfold):
        self.grid = GridSearchCV(estimator = self.clf, param_grid = parameters, cv = Kfold)

    def grid_fit(self, X, Y):
        self.grid.fit(X, Y)

    def grid_predict(self, X, Y):
        self.predictions = self.grid.predict(X)
        print("Precision: {:.2f} % ".format(100*metrics.accuracy_score(Y, self.predictions)
```

In [67]:

```
selected_customers.head()
```

Out[67]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	
0	12347	5	382.52	711.79	558.172000	2790.86	33.948317	10.442659	14.524555	8
1	12348	4	227.44	892.80	449.310000	1797.24	61.983931	38.016069	0.000000	(
2	12350	1	334.40	334.40	334.400000	334.40	60.406699	11.692584	27.900718	(
3	12352	6	144.35	840.30	345.663333	2073.98	66.125517	0.491808	3.370331	14
4	12353	1	89.00	89.00	89.000000	89.00	57.752809	0.000000	19.887640	22

In [68]:

```
columns = ['mean', 'categ_0', 'categ_1', 'categ_2', 'categ_3', 'categ_4' ]
X = selected_customers[columns]
Y = selected_customers['cluster']
```

In [69]:

```
X_train, X_test, Y_train, Y_test = model_selection.train_test_split(X, Y, train_size = 0.8)
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/model_selection/_split.py:202
6: FutureWarning: From version 0.21, test_size will always complement train_
size unless both are specified.
  FutureWarning)
```

In [70]:

```
svc = Class_Fit(clf = svm.LinearSVC)
svc.grid_search(parameters = [{'C':np.logspace(-2,2,10)}], Kfold = 5)
```

In [71]:

```
svc.grid_fit(X = X_train, Y = Y_train)
```

In [72]:

```
svc.grid_predict(X_test, Y_test)
```

```
Precision: 70.78 %
```

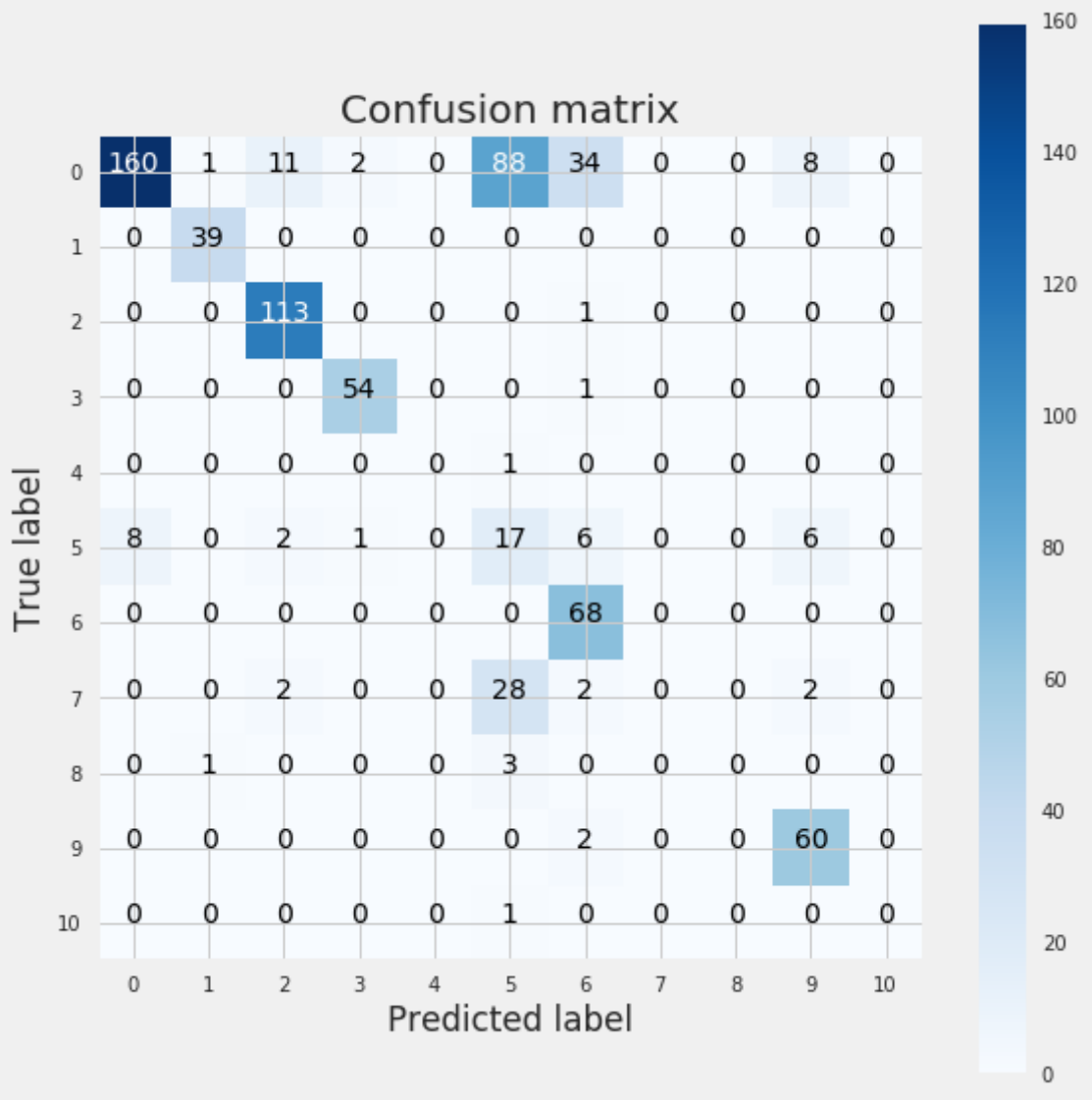
In [73]:

```
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=0)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    #_____________________________________________________
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

In [74]:

```
class_names = [i for i in range(11)]
cnf_matrix = confusion_matrix(Y_test, svc.predictions)
np.set_printoptions(precision=2)
plt.figure(figsize = (8,8))
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize = False, title='Confusion
```

```
Confusion matrix, without normalization
```

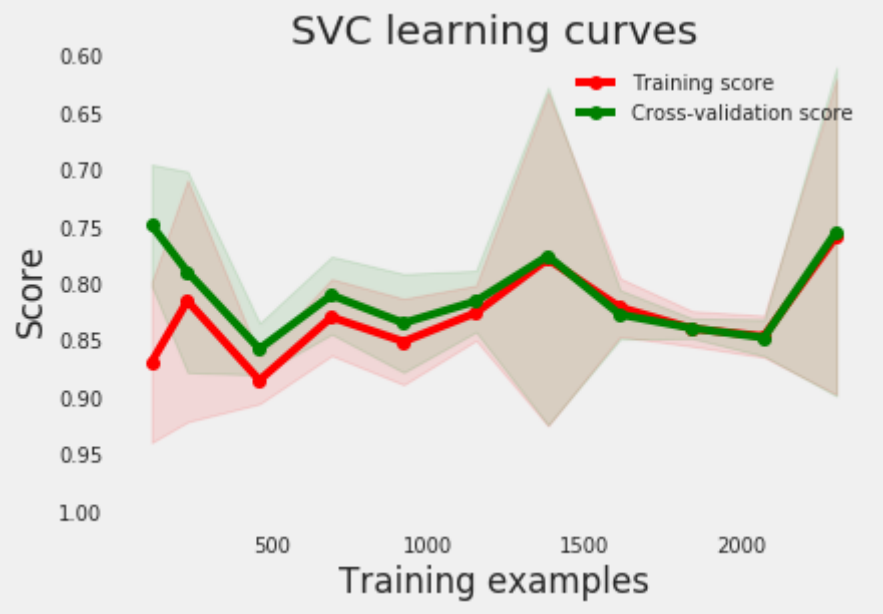


In [75]:

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=-1, train_sizes=np.linspace(.1, 1.0, 10)):
    """Generate a simple plot of the test and training learning curve"""
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, n_jobs=n_jobs, train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g", label="Cross-validation score")

    plt.legend(loc="best")
    return plt
```

In [76]:

```
g = plot_learning_curve(svc.grid.best_estimator_,
                        "SVC learning curves", X_train, Y_train, ylim = [1.01, 0.6],
                        cv = 5,  train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5,
                                                0.6, 0.7, 0.8, 0.9, 1])
```



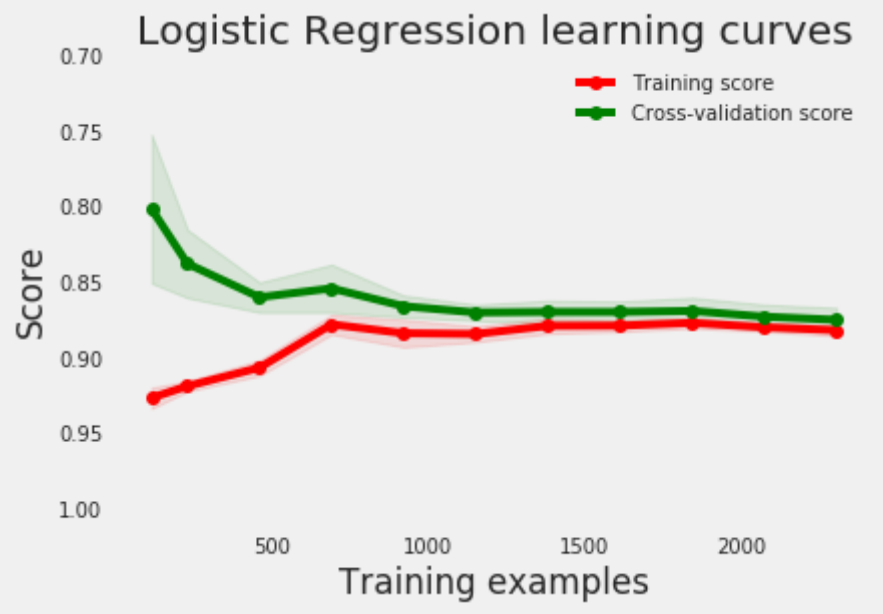
In [77]:

```
lr = Class_Fit(clf = linear_model.LogisticRegression)
lr.grid_search(parameters = [{'C':np.logspace(-2,2,20)}], Kfold = 5)
lr.grid_fit(X = X_train, Y = Y_train)
lr.grid_predict(X_test, Y_test)
```

```
Precision: 86.84 %
```

In [78]:

```
g = plot_learning_curve(lr.grid.best_estimator_, "Logistic Regression learning curves", X_t
                        ylim = [1.01, 0.7], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



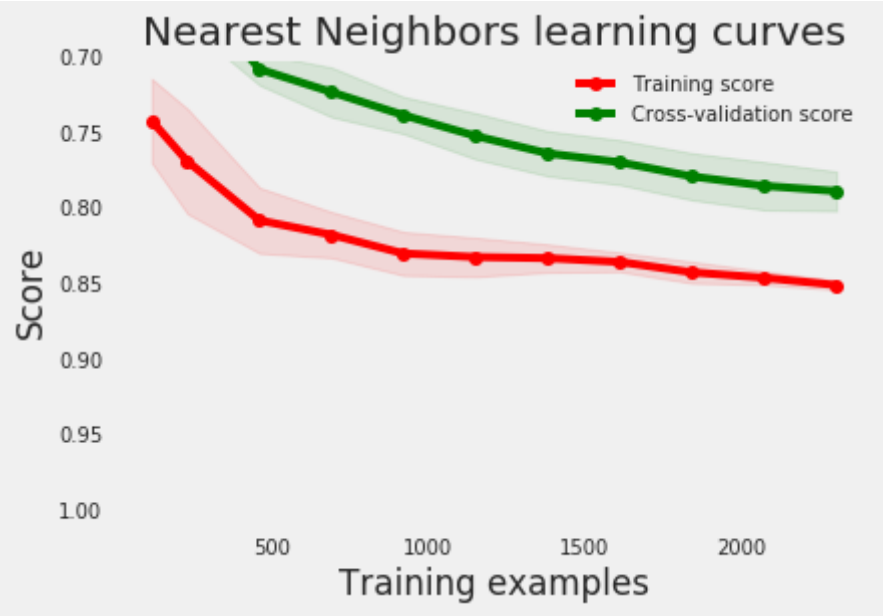
In [79]:

```
knn = Class_Fit(clf = neighbors.KNeighborsClassifier)
knn.grid_search(parameters = [{'n_neighbors': np.arange(1,50,1)}], Kfold = 5)
knn.grid_fit(X = X_train, Y = Y_train)
knn.grid_predict(X_test, Y_test)
```

```
Precision: 81.72 %
```

In [80]:

```
g = plot_learning_curve(knn.grid.best_estimator_, "Nearest Neighbors learning curves", X_tr
                        ylim = [1.01, 0.7], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



Decision Tree

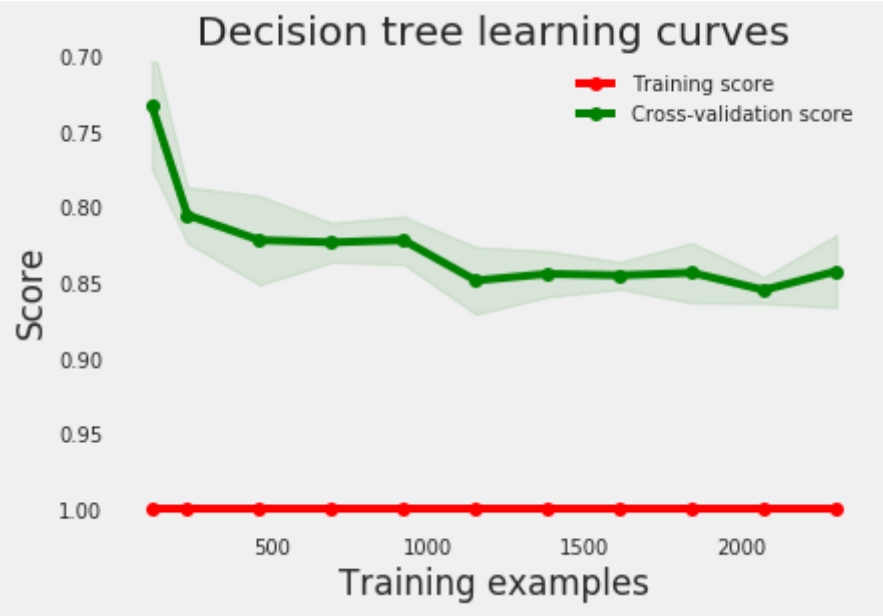
In [81]:

```
tr = Class_Fit(clf = tree.DecisionTreeClassifier)
tr.grid_search(parameters = [{'criterion' : ['entropy', 'gini'], 'max_features' :['sqrt', '
tr.grid_fit(X = X_train, Y = Y_train)
tr.grid_predict(X_test, Y_test)
```

```
Precision: 85.73 %
```

In [82]:

```
g = plot_learning_curve(tr.grid.best_estimator_, "Decision tree learning curves", X_train,
                        ylim = [1.01, 0.7], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



Random Forest

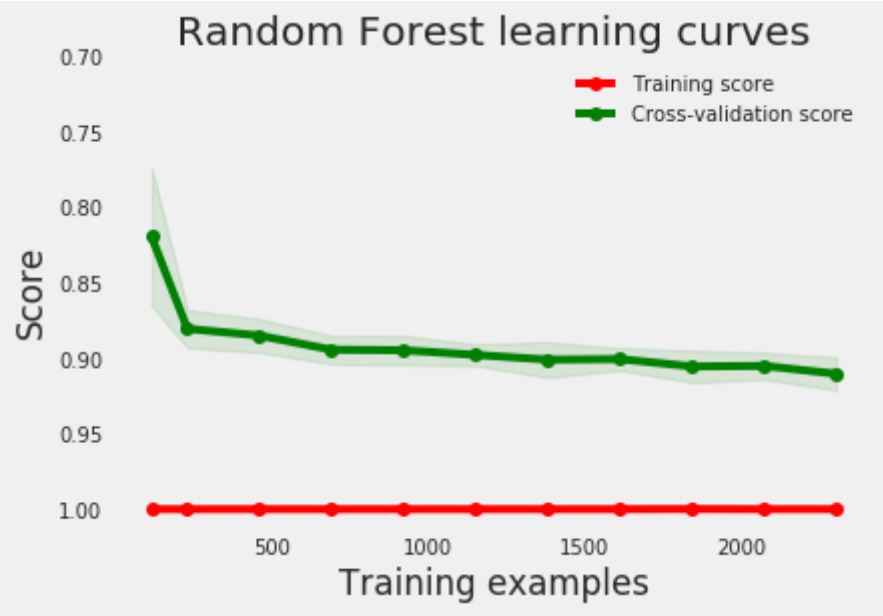
In [83]:

```
rf = Class_Fit(clf = ensemble.RandomForestClassifier)
param_grid = {'criterion' : ['entropy', 'gini'], 'n_estimators' : [20, 40, 60, 80, 100],
              'max_features' :['sqrt', 'log2']}
rf.grid_search(parameters = param_grid, Kfold = 5)
rf.grid_fit(X = X_train, Y = Y_train)
rf.grid_predict(X_test, Y_test)
```

```
Precision: 89.34 %
```

In [84]:

```
g = plot_learning_curve(rf.grid.best_estimator_, "Random Forest learning curves", X_train,
                        ylim = [1.01, 0.7], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



AdaBoost Classifier

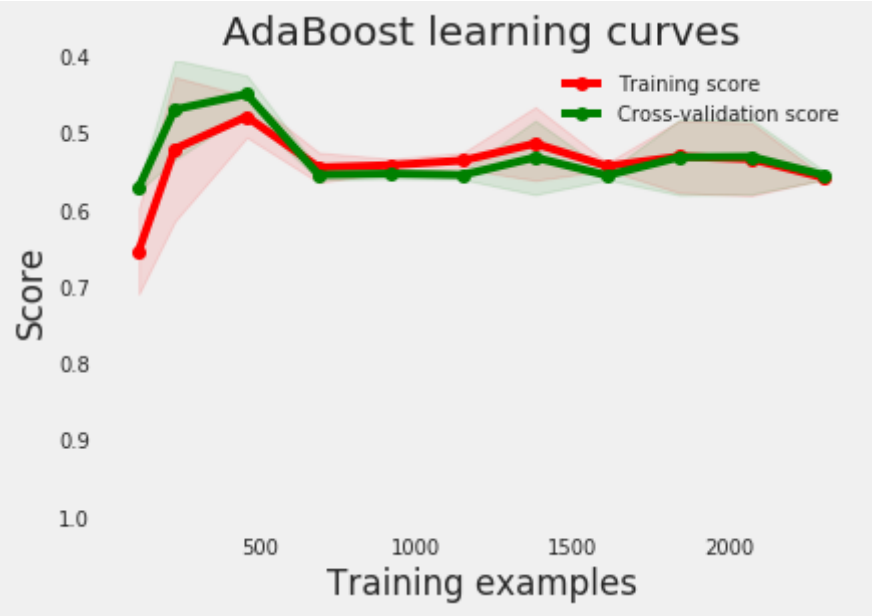
In [85]:

```
ada = Class_Fit(clf = AdaBoostClassifier)
param_grid = {'n_estimators' : [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
ada.grid_search(parameters = param_grid, Kfold = 5)
ada.grid_fit(X = X_train, Y = Y_train)
ada.grid_predict(X_test, Y_test)
```

```
Precision: 55.96 %
```

In [86]:

```
g = plot_learning_curve(ada.grid.best_estimator_, "AdaBoost learning curves", X_train, Y_tr
                        ylim = [1.01, 0.4], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



Gradient Boosting Classifier

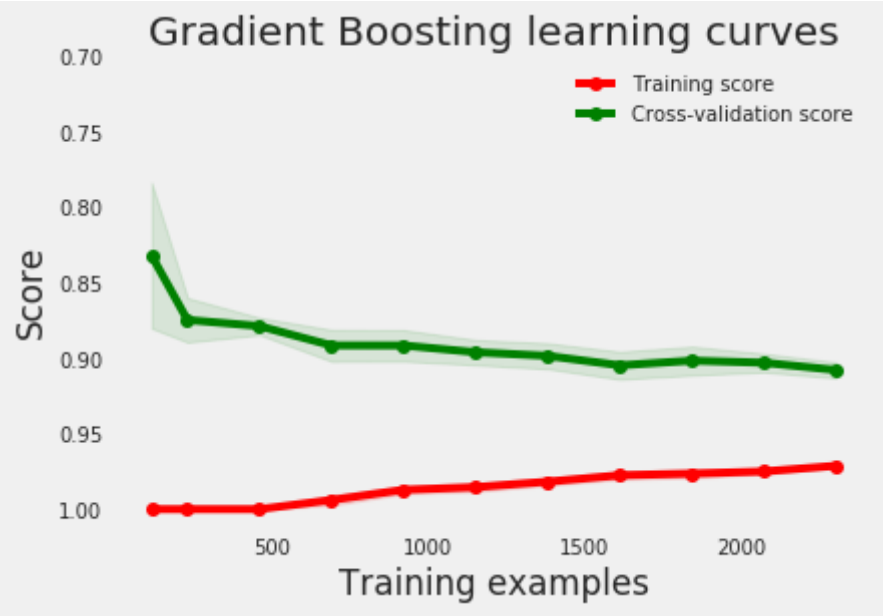
In [87]:

```
gb = Class_Fit(clf = ensemble.GradientBoostingClassifier)
param_grid = {'n_estimators' : [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]}
gb.grid_search(parameters = param_grid, Kfold = 5)
gb.grid_fit(X = X_train, Y = Y_train)
gb.grid_predict(X_test, Y_test)
```

```
Precision: 89.47 %
```

In [88]:

```
g = plot_learning_curve(gb.grid.best_estimator_, "Gradient Boosting learning curves", X_tra
                        ylim = [1.01, 0.7], cv = 5,
                        train_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
```



In [89]:

```
rf_best  = ensemble.RandomForestClassifier(**rf.grid.best_params_)
gb_best  = ensemble.GradientBoostingClassifier(**gb.grid.best_params_)
svc_best = svm.LinearSVC(**svc.grid.best_params_)
tr_best  = tree.DecisionTreeClassifier(**tr.grid.best_params_)
knn_best = neighbors.KNeighborsClassifier(**knn.grid.best_params_)
lr_best  = linear_model.LogisticRegression(**lr.grid.best_params_)
```

In [90]:

```
votingC = ensemble.VotingClassifier(estimators=[('rf', rf_best),('gb', gb_best),
                                                ('knn', knn_best)], voting='soft')
```

In [91]:

```
votingC = votingC.fit(X_train, Y_train)
```

In [92]:

```
predictions = votingC.predict(X_test)
print("Precision: {:.2f} % ".format(100*metrics.accuracy_score(Y_test, predictions)))
```

```
Precision: 90.03 %

/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: D
eprecationWarning: The truth value of an empty array is ambiguous. Returning
False, but in future this will result in an error. Use `array.size > 0` to c
heck that an array is not empty.
  if diff:
```

In [93]:

```
basket_price = set_test.copy(deep = True)
```

In [94]:

```
transactions_per_user=basket_price.groupby(by=['CustomerID'])['Basket Price'].agg(['count',
for i in range(5):
    col = 'categ_{}'.format(i)
    transactions_per_user.loc[:,col] = basket_price.groupby(by=['CustomerID'])[col].sum() /
                                            transactions_per_user['sum']*100

transactions_per_user.reset_index(drop = False, inplace = True)
basket_price.groupby(by=['CustomerID'])['categ_0'].sum()

#_____________________________
# Correcting time range
transactions_per_user['count'] = 5 * transactions_per_user['count']
transactions_per_user['sum']   = transactions_per_user['count'] * transactions_per_user['me

transactions_per_user.sort_values('CustomerID', ascending = True)[:5]
```

Out[94]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2
0	12347	10	224.82	1294.32	759.57	7595.70	25.053649	12.696657	32.343299
1	12349	5	1757.55	1757.55	1757.55	8787.75	52.138488	4.513101	12.245455
2	12352	5	311.73	311.73	311.73	1558.65	60.084047	6.672441	8.735123
3	12356	5	58.35	58.35	58.35	291.75	100.000000	0.000000	0.000000
4	12357	5	6207.67	6207.67	6207.67	31038.35	26.686341	5.089832	14.684737

In [95]:

```
list_cols = ['count','min','max','mean','categ_0','categ_1','categ_2','categ_3','categ_4']
#_____________________________________________________________
matrix_test = transactions_per_user[list_cols].as_matrix()
scaled_test_matrix = scaler.transform(matrix_test)
```

In [96]:

```
Y = kmeans.predict(scaled_test_matrix)
```

In [97]:

```
columns = ['mean', 'categ_0', 'categ_1', 'categ_2', 'categ_3', 'categ_4' ]
X = transactions_per_user[columns]
```

In [98]:

```
classifiers = [(svc, 'Support Vector Machine'),
               (lr, 'Logostic Regression'),
               (knn, 'k-Nearest Neighbors'),
               (tr, 'Decision Tree'),
               (rf, 'Random Forest'),
               (gb, 'Gradient Boosting')]
#_____________________________
for clf, label in classifiers:
    print(30*'_', '\n{}'.format(label))
    clf.grid_predict(X, Y)
```

```
______________________________
Support Vector Machine
Precision: 62.92 %

______________________________
Logostic Regression
Precision: 72.68 %

______________________________
k-Nearest Neighbors
Precision: 67.78 %

______________________________
Decision Tree
Precision: 73.27 %

______________________________
Random Forest
Precision: 75.93 %

______________________________
Gradient Boosting
Precision: 76.05 %
```

In [99]:

```
predictions = votingC.predict(X)
print("Precision: {:.2f} % ".format(100*metrics.accuracy_score(Y, predictions)))
```

```
Precision: 76.83 %

/usr/local/lib/python3.5/dist-packages/sklearn/preprocessing/label.py:151: D
eprecationWarning: The truth value of an empty array is ambiguous. Returning
False, but in future this will result in an error. Use `array.size > 0` to c
heck that an array is not empty.
  if diff:
```