RUTGERS
Rutgers Business School
Newark and New Brunswick

E-Commerce Data Customer Segmentation

Renuka Suhas Madhugiri

Capstone Project

under the direction of

Prof. Meng Qu

Newark, New Jersey

December 2020

ABSTRACT

The author of this project aims to analyze the content of the Ecommerce data by exploring its different attributes and provide insights on the product categories using cluster formation and formatting the data using customer categories. Further the customers have been classified using different classification models and the models have been graded and chosen by comparing the training score and the cross-validation score.

The results obtained for customer behavior using these models have been used to predict the customer category. The use of different machine learning model, algorithms and classifiers were learned during the Business analytics programming, introduction to Data science and multivariate analysis courses during my Master of Information Technology and analytics program.

The work in this project is cumulative representation of my skills and the knowledge obtained during my graduate program and the skills gained with the guidance of my Capstone project supervisor.

The Analysis has been conducted on 2 months of test data and is predicted to have correctly classified about 75% of the customers getting assigned to correct category even though the data is of very small duration and can have various potential shortcomings.

ACKNOWLEDGEMENTS

This Capstone Project has been successfully completed thanks to the guidance and supervision of Professor Meng Qu. I really appreciate the assistance and mentor ship she has provided.

I would also like to thank the Master of Information Technology and Analytics Program, Management Science and Information Systems Department, the Data Mining Lab and all of my previous course professors and colleagues at Rutgers Business School.

My academic, professional, and personal experiences during my time in the Master's program has been enriching and delightful. I hope to apply the technical, theoretical, social and practical components acquired during my graduate study towards my future professional or academic endeavors.

TABLE OF CONTENTS

1. INTRODUCTION

Exploring E-Commerce Data:

The whole world is immersed in data from different sources, hence whenever a customer or a buyer clicks the mouse then that information trail is captured and stored. And this information is used by retailers to attract customers for more purchases. Data science helps retailers discover new ways to understand how to retain their “core” customers rather than merely acquiring new customers.

Customers' needs and likings change over a period of time and every e-commerce business wants to win over the competition by fulfilling the customer demands.

Data science algorithms help businesses understand products, services, processes and customers effectively.

With this dataset we will be answering certain questions like

1. Which countries have the highest sales?
2. Is there any certain time when sales are highest?
3. Are there any most sold items?

2. DATASET

This dataset is from [The UCI Machine Learning Repository] and contains actual transactions for a UK based retailer from year 2010 to 2011 and it can be found by title "Online Retail".

The ecommerce data has 8 columns and 541910 records of transactions.While looking at the number of null values in the dataframe, it is interesting to note that almost 25% of the entries are not assigned to a particular customer.

The initial dimension of the data is (541909,8) and after removing null values it is (406829,8). It has details for particular transactions like InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, Country.

Detailed description of the field is as follows

Column Name	Data Type	Description
InvoiceNo	character	Invoice Number - unique number generated for every new sell
StockCode	character	Stock code - item code
Description	character	Item description
Quantity	numeric	Item quantity for particular invoice
InvoiceDate	character	Date of the invoice
UnitPrice	numeric	Per unit price of the item
CustomerID	numeric	Customer number
Country	character	Country where the item is being sold

3. PROBLEM STATEMENT

E-Commerce industry:

1. E-commerce industry has evolved significantly in decision making over time. Earlier they used to do basket analysis for recommendation, today we have customer specific predictive algorithms being executed.

2. Recommendation systems and other technology is now executed in seconds, which make them even more effective.

3. We have big data environment and no more see data stored in CSV formats. To churn millions of activities of billions of customers, we need parallelization of processes. Everything is done seamlessly by E-Commerce industry and customer are mostly unaware about these processes.

   These recommendations are being used by amazon and various ecommerce websites inspired me to dig deeper into the customer recommendations and hence I wanted to implement a classification with customer segmentation on the E-commerce data.

4. METHODOLOGY & RELEVANT WORK :

For any data science or machine learning project we need to follow sequence of steps which start with data cleaning or preparing the data for exploration as the dataset has many impurities like null values, duplicate and invalid entries and in order for the data to be ready for analysis it must be cleaned.

4.1 Data Cleaning:

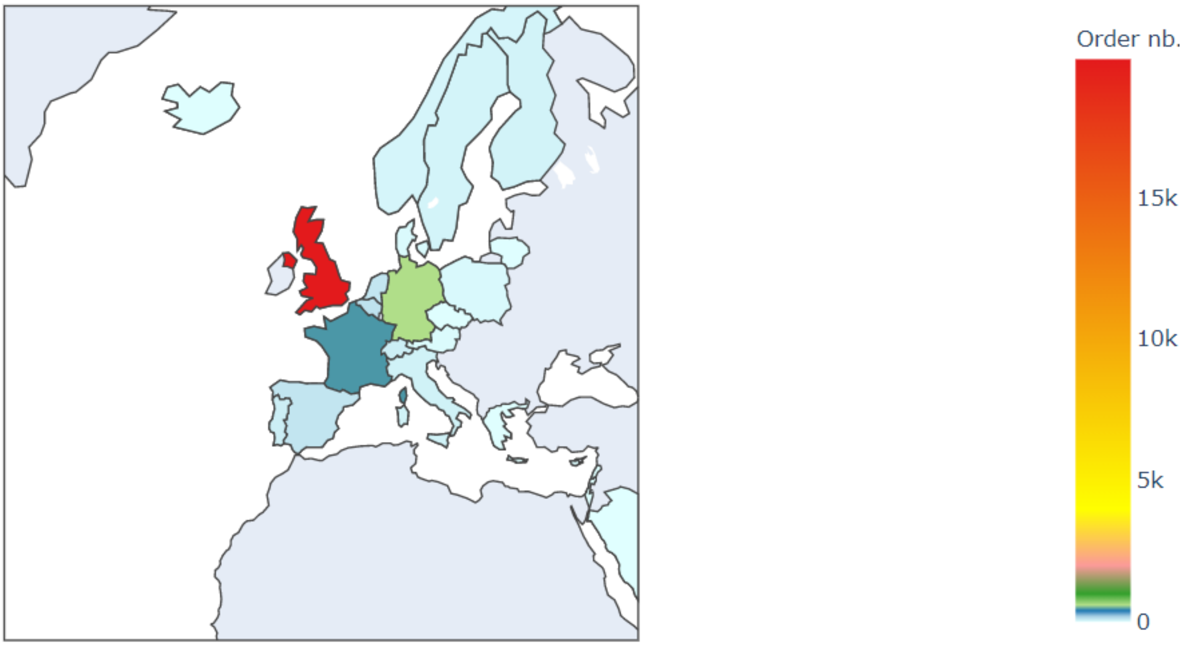
For this data CustomerID had around 25% of the null values and Description had some null values as well so cleaned the data and got rid of those using dropna method and later deleted the duplicate entries using drop_duplicates() function.

4.2 Explanatory Data Analysis:

After data cleaning next step is exploring the data and in that first is visualization of it.

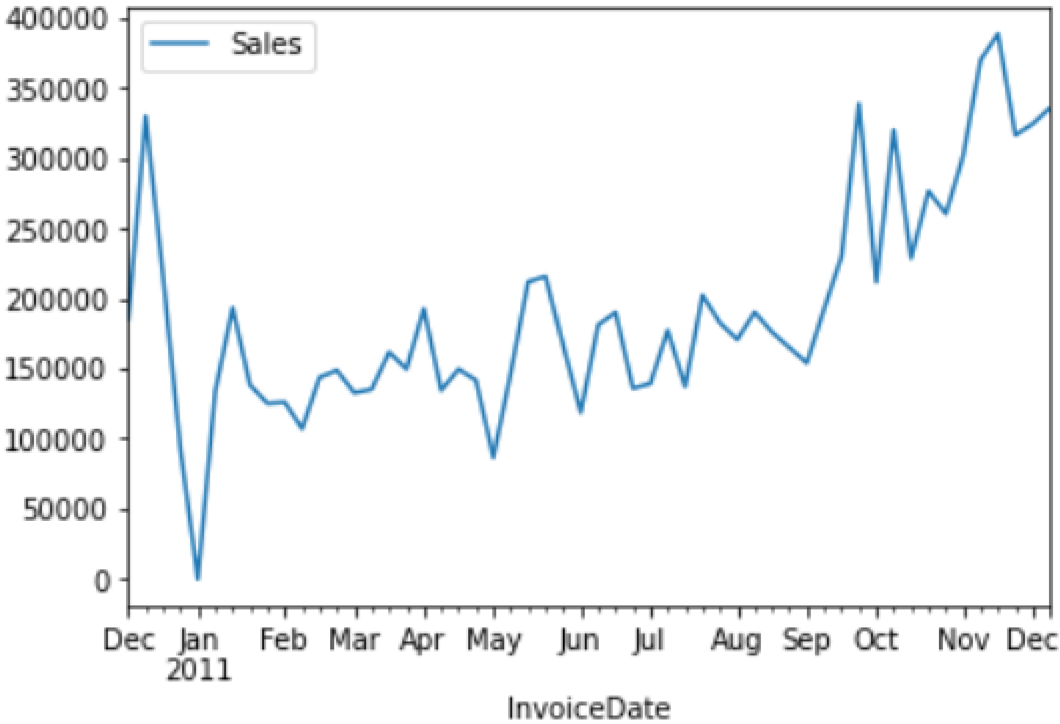
As we already know the data is from UK e-commerce retailer the maximum sell must be from UK and for validation plotted it on a map as we can see in the below image maximum of the items are sold in UK and very less out of UK.

Number of orders per country



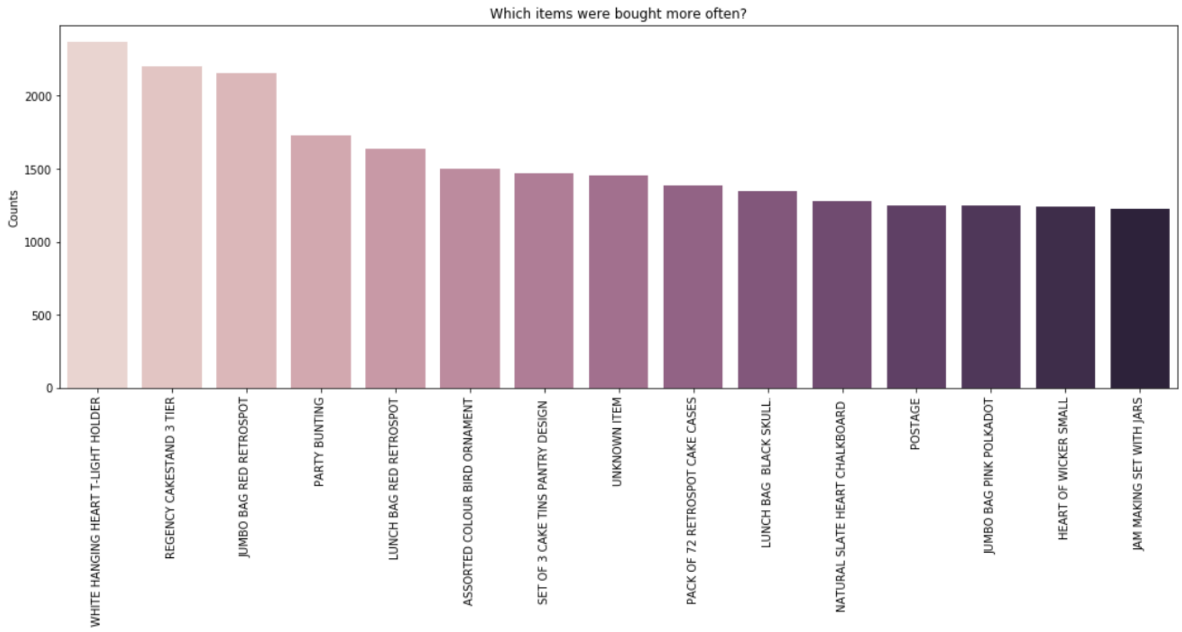
This answers our 1st question which country has highest number of sales and the answer is UK.

The second question can also be answered with visualization. Ia there a time when we have highest sell



As we can see in the below image there is a peak at around end of November which is the Thanksgiving period and that is the period when we have highest sales.

Also we have another question if we have any most sold items



We have the above graph which states the most sold or bought items

Further to cleaning and visualizations I explored the cotents in variables by forming clusters and grouping the variables using different categories like

Determining the number of products bought in each transaction as we have data which is naturally grouped in from of products hence regrouping to determine the products bought in each transaction.

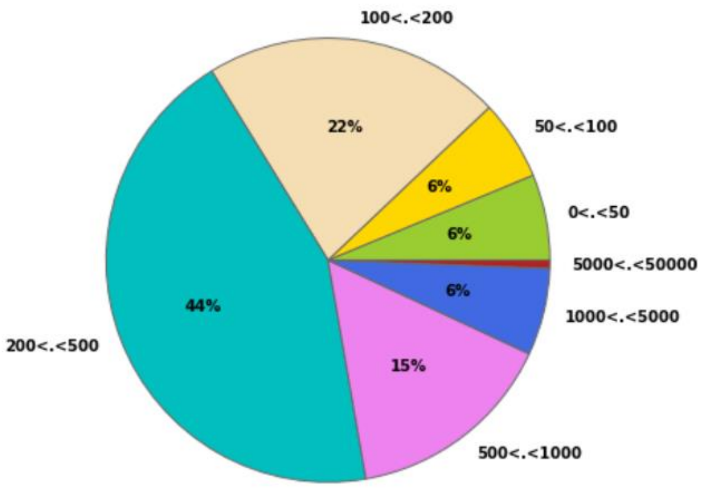
With this grouping I found out that many transactions are from cancelled orders and the data has about 16% of the transactions which correspond to cancellation.

The cancellation do not necessarily correspond to the transaction made in the dataset period.

Some of the cancellation do not have any accounting or counterpart for the data as the sale might have been made before December 2010.

Hence I created 2 cases with entry_to_remove are the cancellations and doubtfull_entry is then one which has no previous record which is around 0.2% and 1.4% of the total transactions.

Later to cancellations and product grouping with the invoice numbers I observed how the purchases were divided according to the total price



As we can see here most of the orders are more than 200 Euros which is almost 65% of the purchases done.

Another kind of grouping I did was with the Description of the product with unique values in description and then took the words which appeared more than 13 times And then appended 6 more columns with the price range for this I used One-hot-encoding principle and split the data using the mean of unit price for the product and the description.

After the grouping with price ranges I grouped the products in f=different classes and formed clusters using k-means for that I used the Kmodes package and calculated the silhouette score for the clusters and found that when the number of clusters is more than 5 the silhouette score is highest that is 0.147.

The clustering gave the distribution of products

```
0    1009
1     964
3     673
2     626
4     606
dtype: int64
```

We can visualize the wordcloud for the clusters as it will give us the type of clusters formed

As we can see above the cluster n1 has Christmas stuff grouped together and cluster n4 has the jewelry like bracelet ring etc. grouped together.

In order to check for the distinct behavior of clusters I performed PCA on the data used for clustering. And found that For 90% of variance in the data we need at least 100 components.

4.3 Customer Categorization:

Earlier we grouped the products in 5 clusters now for rest of the analysis we must have categories of the product and hence created a variable categ_product which groups the data in 5 categories from 0 to 4 using the the unique description we used while clustering and the cluster data.Later this data is grouped per order and distributed them among the 5 categories.

For validation, training and testing of data it must be split into train and test data and hence splitting the data in test and train so the first 10 months are used for training while the last 2 months for testing and validations.

I also found the unique transactions as in the customers who just bought once and It came out to be 1444 out of 3610 transactions that is 40% of the total transactions.

As per the earlier clustering the silhouette score was the best for 11 number of clusters that is 0.210

We can observe the distribution of clients/customers in clusters as follows.

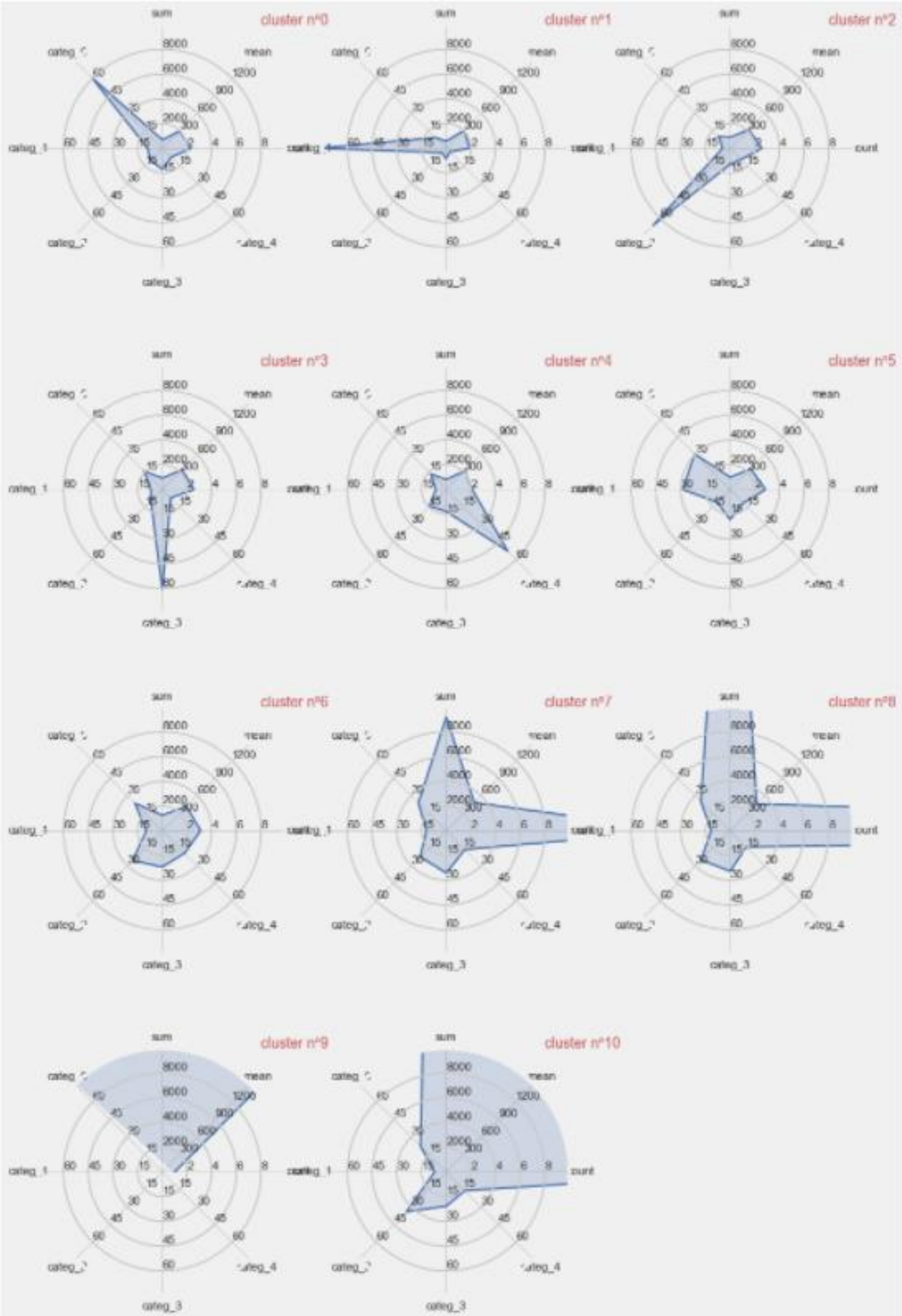
:

	4	5	3	1	9	6	10	0	8	7	2
nb. de clients	1152	962	399	277	276	201	174	151	9	8	1

I performed PCA on this again and observed the clusters formed and we can observe that we have disjoint distinct clusters.

Then the clustered customers were grouped in each category and amount spent on the product and those selected customers were represented with the radar diagram representing the amount spent by customer in each category and then the total amount spent.

We can observe the diagram below and comment that first 5 clusters explain the purchasing of the different categories of the product. Other clusters differ with basket_average that is mean, amount spent by the customers or the number of visits by the customer.



cluster n°0
cluster n°1
cluster n°2
cluster n°3
cluster n°4
cluster n°5
cluster n°6
cluster n°7
cluster n°8
cluster n°9
cluster n°10
sum
mean
categ_1
categ_3

4.4 Customer classification for different classification models:

The main objective of the project is the classification the customer and for that we need to have customer categories which we have already created splitting the data in train and test.

I ran various classification algorithms for this data they are as follows:

1. Support Vector Machine Classifier (SVC)
2. Logistic Regression
3. K-nearenst Neighbours
4. Decision tree
5. Random Forest
6. AdaBoost Classifier
7. Gradient Boosting Classifier

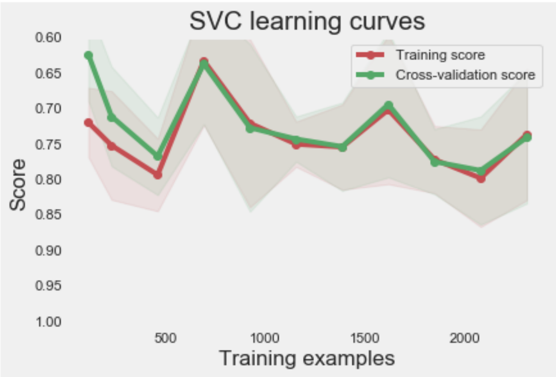
These are explained further in Experiment section of the research paper.

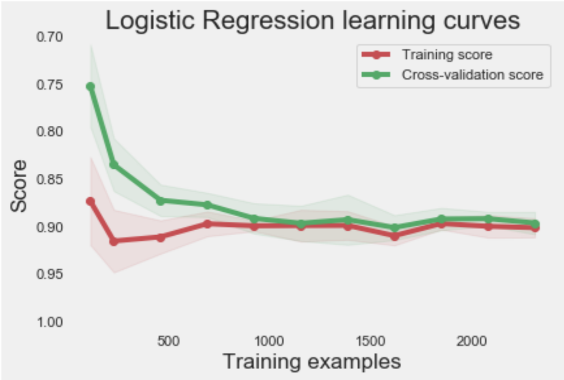
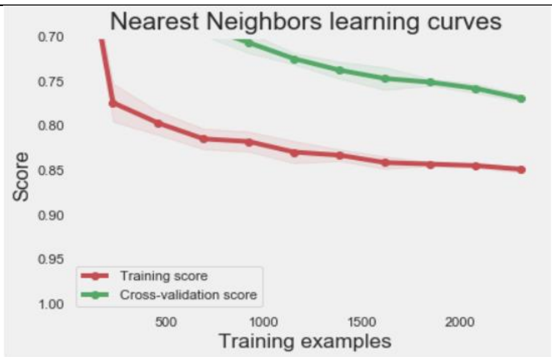
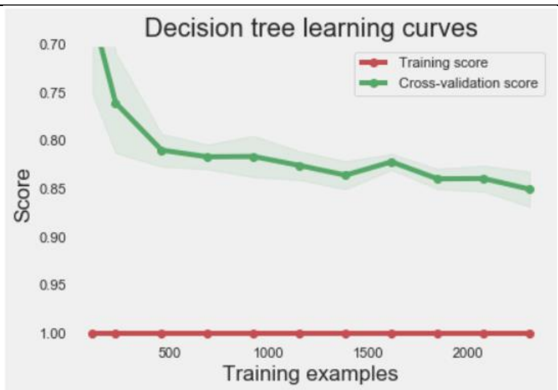
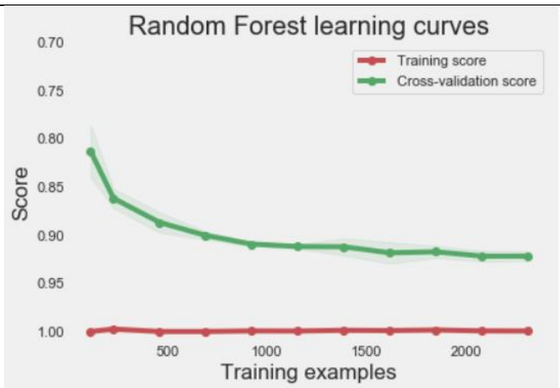
5. EXPERIMENT

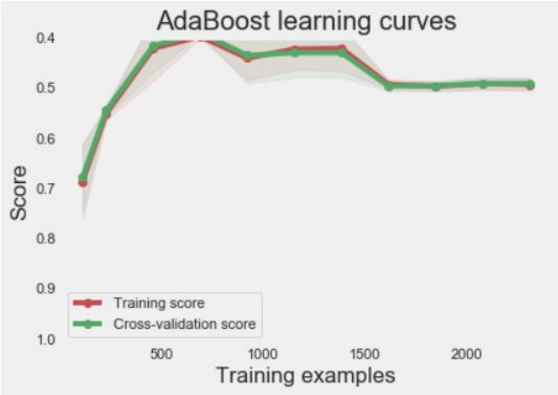
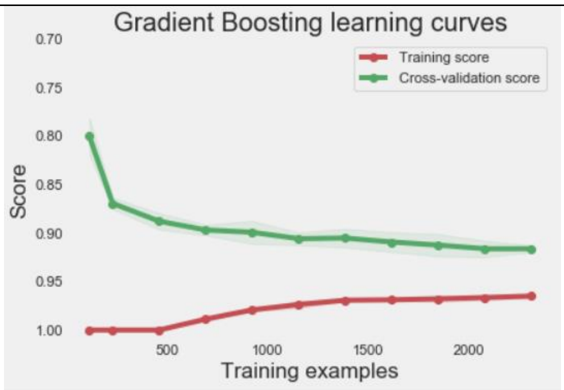
Continuing the data modeling and classification from the methodology part in the experiment section. I performed different types on classification on the cleaned data and later compared the results from each classifier and selected the best amongst them for further predictions.

In order to test the quality of the fit we draw learning and the cross validation curve, these curves help us understand the over or under fitting of the implemented models

The following table has the precision score, the CV and training graph with the observations obtained from each of the classifications performed.

Classifier Type	Precision Score (in %)	Graph for training and CV curve	Observations
SVC (Support Vector Machine Classifier)	71.33		The training and crossvalidation curve converge towards the end hence we have low varience model and it doesn't overfit the dataalso the accuracy of trainig cur e is accuarte and hence dat ais niot underfitted

Logistic Regression	90.17		Similar to SVC these curves also converge towards the end and hence are not overfitting the data
K-nearest Neighbor	76.73		For this the curves appear to converge future in the data and hence fits the data and doent over or underfit the data
Decision Tree	86.29		The random forests suffers from high variance and low bias and hence overfits the training data.
Random Forest	91.41		Similar to random forest the decision tree algorithm also suffers from high variance and low bias and hence overfits the training data.

AdaBooster Classifier	48.89		Similar to SVC and logistic regression these curves also converge towards the end and hence are not overfitting the data
Gradient Boosting Classifier	91.55		Similar to KNN these curves appear to converge future in the data and hence fits the data and doesnt over or underfit the data

To have the best possible results we can combine the results from different classifiers. By selecting the customer category indicated by most of the classifiers we can achieve the best results.

For this I have used the VotingClassifier method from sklearn package by adjusting the best scored from all the classifiers.

I have merged the results from Random Forest, Gradient Boosting, KNN and then train the classifier and finally we have the prediction with precision of 92.11%.

5.1 TESTING DATA:

We test the trained data on the last 2 months which we kept as the testing data. For this the test data is formatted in the same form as that of the training data. And this data contains the buying behavior of the customer and is like the categories we created for the training data earlier in the methodology section.

The category of the customer is found using the Kmeans algorithm which calculate sthe distance from the centroids for the 11 customer classes which we created with 11 clusters and the smallest distance is calculated for all the categories.

The following are the results we obtain from the classifiers which we trained earlier.

```
______________________________
Support Vector Machine
Precision: 60.86 %
______________________________
Logostic Regression
Precision: 75.34 %
______________________________
k-Nearest Neighbors
Precision: 64.27 %
______________________________
Decision Tree
Precision: 72.37 %
______________________________
Random Forest
Precision: 75.69 %
______________________________
Gradient Boosting
Precision: 75.73 %
```

For the validation we find the precision for predictions of the combined classifier for random forest, Gradient Boosting and KNN as it provides a bit better predictions.

The precision obtained is 76.28%

6. CONCLUSION

This Research paper is based on the notebook which has details about the E-commerce dataset which provides the transaction details for different customers at different times at a particular date, the data has almost 4000 customers. I developed the classifier which tells about the number of visits a customer can make from his first visit to site.

I grouped customers in 5 categories of the products and classified the customers based on their buying behavior over a span of 10 month. furthermore, I classified the customers in 11 categories using 5 variables mean and categories from the amount spent by customer for each product.

Ultimately the predictions are tested over the test data of 2 months with these predictions and validations we found that 75% of the customers were assigned to the right classes and hence the performance seems to be satisfactory with the possible shortcomings of the classifier and the models.

We can check for the seasonality for the dataset if we have more data entries in the dataset.

8. APPENDIX

```
# # Customer segmentation

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 wordcloud import WordCloud, STOPWORDS
from sklearn.ensemble import AdaBoostClassifier
from sklearn.decomposition import PCA
from IPython.display import display, HTML
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)
get_ipython().run_line_magic('matplotlib', 'inline')


# read data
df_initial = pd.read_csv(r'C:\Users\rmadh\OneDrive\Desktop\Lecture_Notes\Capstone\ecom_data.csv',
encoding = 'ISO-8859-1')
print('Dataframe dimensions:', df_initial.shape)
#______
df_initial['InvoiceDate'] = pd.to_datetime(df_initial['InvoiceDate'])
#____________________________________________________________________
# Inforamtion about data
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 values (no)'}))
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)
#__________________
# first 5 lines
display(df_initial[:5])

df_initial.dropna(axis = 0, subset = ['CustomerID'], inplace = True)
print('Dataframe dimensions:', df_initial.shape)

# info about 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 values (nb)'}))
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)

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

temp = df_initial[['CustomerID', 'InvoiceNo', 'Country']].groupby(['CustomerID', 'InvoiceNo',
'Country']).count()
temp = temp.reset_index(drop = False)
countries = temp['Country'].value_counts()
print('different countries  in the data: {}'.format(len(countries)))


#result on a chloropleth map:

# In[11]:


data = dict(type='choropleth',
locations = countries.index,
locationmode = 'country names', z = countries,
text = countries.index, colorbar = {'title':'Order nb.'},
colorscale=[[0, 'rgb(224,255,255)'],
            [0.01, 'rgb(166,206,227)'], [0.02, 'rgb(31,120,180)'],
            [0.03, 'rgb(178,223,138)'], [0.05, 'rgb(51,160,44)'],
            [0.10, 'rgb(251,154,153)'], [0.20, 'rgb(255,255,0)'],
            [1, 'rgb(227,26,28)']],
reversescale = False)
#_________________________
layout = dict(title='Number of orders per country',
geo = dict(showframe = True, projection={'type':'mercator'}))
#_______________
choromap = go.Figure(data = [data], layout = layout)
iplot(choromap, validate=False)


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'])

# Now I will determine the number of products purchased in every transaction:

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

2.2.1 Order Cancellations

nb_products_per_basket['order_canceled'] = nb_products_per_basket['InvoiceNo'].apply(lambda x:int('C' in x))
display(nb_products_per_basket[:5])
#______________________________________________________________________________________________
n1 = nb_products_per_basket['order_canceled'].sum()
n2 = nb_products_per_basket.shape[0]
print('Number of orders canceled: {}/{} ({:.2f}%) '.format(n1, n2, n1/n2*100))

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

```
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

df_check = df_initial[(df_initial['Quantity'] < 0) & (df_initial['Description'] != 'Discount')][
                                 ['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


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  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


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


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'] != 'D')]
print("nb of entries to delete: {}".format(remaining_entries.shape[0]))
remaining_entries[:5]


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

list_special_codes = df_cleaned[df_cleaned['StockCode'].str.contains('^[a-zA-Z]+',
regex=True)]['StockCode'].unique()
list_special_codes


for code in list_special_codes:
    print("{:<15} -> {:<30}".format(code, df_cleaned[df_cleaned['StockCode'] ==
code]['Description'].unique()[0]))

#Basket Price

# In[24]:


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

# Sum of Purchases
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['TotalPrice'].sum()
basket_price = temp.rename(columns = {'TotalPrice':'Basket Price'})
#_____________________
# Order Date/ Invoice Date
df_cleaned['InvoiceDate_int'] = df_cleaned['InvoiceDate'].astype('int64')
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate_int'].mean()
df_cleaned.drop('InvoiceDate_int', axis = 1, inplace = True)
basket_price.loc[:, 'InvoiceDate'] = pd.to_datetime(temp['InvoiceDate_int'])
#_______________________________________
# Significant entry selection:
basket_price = basket_price[basket_price['Basket Price'] > 0]
basket_price.sort_values('CustomerID')[:6]

# Purchase Statement
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'].count()
    count_price.append(val)
#_____________________________________________
# Number of purchases/order amount representation
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 != 0]
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, "breakdown of the order amounts", ha='center', fontsize = 18);
```

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

def keywords_inventory(dataframe, colonne = 'Description'):
    stemmer = nltk.stem.SnowballStemmer("english")
    keywords_roots  = dict()  # collect the words
    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("Nb of keywords in variable '{}': {}".format(colonne,len(category_keys)))
    return category_keys, keywords_roots, keywords_select, count_keywords


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


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


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

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[31]:


liste = sorted(list_products, key = lambda x:x[1], reverse = True)
#_______________________________
plt.rc('font', weight='normal')
fig, ax = plt.subplots(figsize=(7, 25))
y_axis = [i[1] for i in liste[:125]]
x_axis = [k for k,i in enumerate(liste[:125])]
x_label = [i[0] for i in liste[:125]]
plt.xticks(fontsize = 15)
plt.yticks(fontsize = 13)
plt.yticks(x_axis, x_label)
plt.xlabel("Nb. of occurences", fontsize = 18, labelpad = 10)
ax.barh(x_axis, y_axis, align = 'center')
ax = plt.gca()
ax.invert_yaxis()
#_______________________________________________________________________________________
plt.title("Words occurence",bbox={'facecolor':'k', 'pad':5}, color='w',fontsize = 25)
plt.show()

#product categories

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('Selected Words:', len(list_products))


# ____
#Data encoding

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

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])
    label_col.append(col)
    X.loc[:, col] = 0

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

```
    X.loc[i, label_col[j-1]] = 1


print("{:<8} {:<20} \n".format('gamme', 'nb. produits') + 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()))


# Clusters of products

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_avg)

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)
    print("For n_clusters =", n_clusters, "The average silhouette_score is :", silhouette_avg)


pd.Series(clusters).value_counts()

def graph_component_silhouette(n_clusters, lim_x, mat_size, sample_silhouette_values, clusters):
    plt.rcParams["patch.force_edgecolor"] = True
    plt.style.use('fivethirtyeight')
    mpl.rc('patch', edgecolor = 'dimgray', linewidth=1)
    #____________________________
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(8, 8)
    ax1.set_xlim([lim_x[0], lim_x[1]])
    ax1.set_ylim([0, mat_size + (n_clusters + 1) * 10])
    y_lower = 10
    for i in range(n_clusters):
        #___________________________________________________________________________________
        # Aggregate the silhouette scores for samples belonging to cluster i, and sort them
        ith_cluster_silhouette_values = sample_silhouette_values[clusters == i]
        ith_cluster_silhouette_values.sort()
        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i
        cmap = cm.get_cmap("Spectral")
        color = cmap(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper), 0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.8)
        #____________________________________________________________________
        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.03, y_lower + 0.5 * size_cluster_i, str(i), color = 'red', fontweight = 'bold',
            bbox=dict(facecolor='white', edgecolor='black', boxstyle='round, pad=0.3'))
        #______________________________________
```

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


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, clusters)

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()))

#_______________________________________________________________________________
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 nº{}'.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)


# Principal Component Analysis

pca = PCA()
```

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

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 = 'g',
           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);

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

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 = (15,8))
increment = 0
for ix in range(4):
    for iy in range(ix+1, 4):
        increment += 1
        ax = fig.add_subplot(2,3,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', facecolor = 'lightgrey',
           shadow = True, frameon = True, framealpha = 1,
           fontsize = 13, bbox_transform = plt.gcf().transFigure)

plt.show()

# Customer categories
```

```
corresp = dict()
for key, val in zip (liste_produits, clusters):
    corresp[key] = val
#___________________________________________________________________________
df_cleaned['categ_product'] = df_cleaned.loc[:, 'Description'].map(corresp)


# In[49]:


df_cleaned.head()


# In[50]:


df_cleaned.tail()
#Product groups

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']][:5]


# Sum of Purchases
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['TotalPrice'].sum()
basket_price = temp.rename(columns = {'TotalPrice':'Basket Price'})
#____________________________________________________________
# percentage of price 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 Invoice/order
df_cleaned['InvoiceDate_int'] = df_cleaned['InvoiceDate'].astype('int64')
temp = df_cleaned.groupby(by=['CustomerID', 'InvoiceNo'], as_index=False)['InvoiceDate_int'].mean()
df_cleaned.drop('InvoiceDate_int', axis = 1, inplace = True)
basket_price.loc[:, 'InvoiceDate'] = pd.to_datetime(temp['InvoiceDate_int'])
#______________________________________
# Significant entry selection:
basket_price = basket_price[basket_price['Basket Price'] > 0]
basket_price.sort_values('CustomerID', ascending = True)[:5]

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


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)

# No. of visists by the customer and its effect on the basket amount
```

```
transactions_per_user=basket_price.groupby(by=['CustomerID'])['Basket
Price'].agg(['count','min','max','mean','sum'])
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]

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

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

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]

n1 = transactions_per_user[transactions_per_user['count'] == 1].shape[0]
n2 = transactions_per_user.shape[0]
print("nb. of unique customers: {:<2}/{:<5} ({:<2.2f}%)".format(n1,n2,n1/n2*100))
```

Customer Category creation

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

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

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 = 'g',
            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);

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('score de silhouette: {:<.3f}'.format(silhouette_avg))

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

pca = PCA(n_components=6)
matrix_3D = pca.fit_transform(scaled_matrix)
mat = pd.DataFrame(matrix_3D)
mat['cluster'] = pd.Series(clusters_clients)
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:'darkcyan', 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', facecolor = 'lightgrey',
           shadow = True, frameon = True, framealpha = 1,
           fontsize = 13, bbox_transform = plt.gcf().transFigure)

plt.tight_layout()


sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
#____________________________________
# define individual silouhette scores
sample_silhouette_values = silhouette_samples(scaled_matrix, clusters_clients)
#_________________
# graph
graph_component_silhouette(n_clusters, [-0.15, 0.55], len(scaled_matrix), sample_silhouette_values,
clusters_clients)

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

```
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')

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']])

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)


# view of cluster:

# In[75]:


fig = plt.figure(figsize=(10,12))

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], [0.01, 75], [0.01, 75]]
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])
    radar = RadarChart(fig, location, sizes, attributes, ranges)
    radar.plot(data, color = 'b', linewidth=2.0)
    radar.fill(data, alpha = 0.2, color = 'b')
    radar.title(title = 'cluster nº{}'.format(index[ind]), color = 'r')
    ind += 1


# Classification
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[77]:


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


# split to train and test sets:

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


#Support Vector Machine Classifier (SVC)
#  instance of the `Class_Fit` class and then call` grid_search()`. provide as parameters:
# - the hyperparameters for which I will seek an optimal value
# - the number of folds to be used for cross-validation

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

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

svc.grid_predict(X_test, Y_test)


#Confusion matrix
def plot_confusion_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):
    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')

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 matrix')


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

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])

#Logistic Regression

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)

g = plot_learning_curve(lr.grid.best_estimator_, "Logistic Regression learning curves", X_train, Y_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])


#k-Nearest Neighbors
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)
```

```
g = plot_learning_curve(knn.grid.best_estimator_, "Nearest Neighbors learning curves", X_train, Y_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])


# Decision Tree

tr = Class_Fit(clf = tree.DecisionTreeClassifier)
tr.grid_search(parameters = [{'criterion' : ['entropy', 'gini'], 'max_features' :['sqrt', 'log2']}], Kfold = 5)
tr.grid_fit(X = X_train, Y = Y_train)
tr.grid_predict(X_test, Y_test)

g = plot_learning_curve(tr.grid.best_estimator_, "Decision tree learning curves", X_train, Y_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

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)

g = plot_learning_curve(rf.grid.best_estimator_, "Random Forest learning curves", X_train, Y_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

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)

g = plot_learning_curve(ada.grid.best_estimator_, "AdaBoost learning curves", X_train, Y_train,
                    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

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)

g = plot_learning_curve(gb.grid.best_estimator_, "Gradient Boosting learning curves", X_train, Y_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])
# Comarison of Classifiers

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_)

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


# Training the classifier
votingC = votingC.fit(X_train, Y_train)


# prediction for model:

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

# prediction testing
basket_price = set_test.copy(deep = True)

transactions_per_user=basket_price.groupby(by=['CustomerID'])['Basket
Price'].agg(['count','min','max','mean','sum'])
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['mean']

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


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)

Y = kmeans.predict(scaled_test_matrix)


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

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)


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

Comclusion:
In research paper

7. REFERENCE

1. https://www.dataquest.io/blog/learning-curves-machine-learning/

2. https://www.analyticsvidhya.com/blog/2015/08/role-analytics-e-commerce-industry/

3.https://www.analyticsvidhya.com/blog/2015/08/role-analytics-e-commerce-industry/

4. https://towardsdatascience.com/data-science-for-e-commerce-with-python-a0a97dd7721d

5.https://scikitlearn.org/stable/auto_examples/model_selection/plot_learning_curve.html#sphx-glr-%20self-examples-model-selection-pad-learning-curve-py

6. https://www.kaggle.com/carrie1/ecommerce-data