GROUPING CUSTOMERS USING CLASSIFICATION ALGORITHMS

***Ravi teja Yamsani*[1]**

Student, Department of Computer Science and Engineering, Kalasalingam Academy of Research and Education, Anand Nagar, Krishnankoil, India

Dr. A. Saravanan

Associate Professor, Department of Computer Science and Engineering, Kalasalingam Academy of Research And Education, Anand Nagar, Krishnankoil, India

***Abstract* -** Meeting of many competitors at once entrepreneurs have created a major conflict between competitors businesses to find new customers and keep old ones. As a result formerly, the need for unique customer service becomes relevant regardless of the size of the business. In addition, the ability of any business to understand the needs of each customer will supply a larger customer support for the provision of targeted customer services and development customized customer service plans. This understanding possible through systematic customer service. Each part has customers who share similar market features. Big data ideas and machine learning have encouraged greater acceptance of automatic customer segregation methods in favor traditional market statistics that often do not work there the customer base is very large. Customer segregation is important in customer relationship management software. The most common way to differentiate one customer from another is to develop a customer group according to their interests and factors like demographic factors, Psychological factors, Behavioral factors, and geographical factors . In this work, the company's hand-separated customer data is analyzed. As machine learning methods are useful for solving problems with data management and data processing, the solution is searched within machine learning methods. The different classification methods are used to differentiate the people into different groups and help with suitable solutions. For increasing the efficiency and precision we used all the classification algorithms as a ensemble algorithm and created a model.

I. INTRODUCTION

From many years, Due to the rise in competitive nature among the businesses and the use of the past data has motivated in implementation of data mining techniques for finding the critical and useful insights from the information. Data mining procedure is helpful for extraction of the logical information from the dataset of the company and project in a human understanding manner such that human can take required pre requisite to solve the problem.

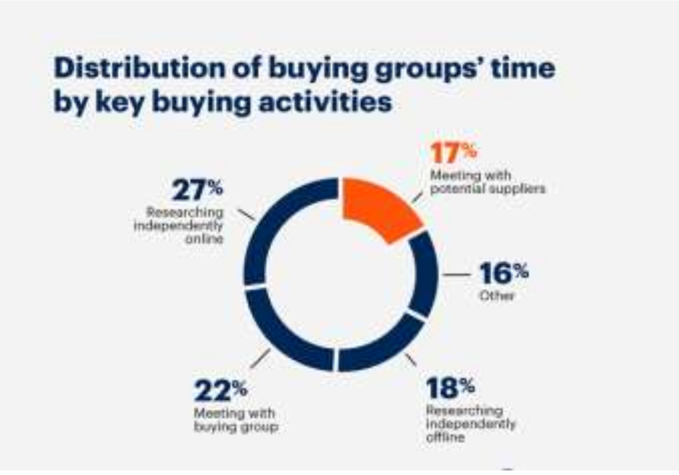


Figure: Gartner company proves that segregating of customers brings 2nd highest sales

II. LITERATURE REVIEW

In literature survey we discussed various machine learning techniques have been proposed by the scientists for identification and grouping the customers into different clusters. This research study shows some existing machine learning based techniques in order to explain importance of the implemented work. The market separation or grouping is based on only few factors for determining the customer group. Over the years, the commercial world has become more competitive, as organizations such as these have to meet the needs and desires of their customers, attract new customers, and thus improve their businesses. For amazon, flip kart, and Netflix the customer grouping is done by item to item and only built by the data received by their website. All their systems algorithms are confidential and not shared with the all but only the basis of their fundamental are shared for privacy concern of the customers. Few data scientists did the project using KNN algorithm for separation but the data considered for the project is likely small and the accuracy is about 78.3%.. Grouping also includes following factors like Demographic, geographic, technographic and behavioural segmentation. Data collection is part of the research in all fields of study including natural and social sciences, people and businesses. The purpose of all data collection to obtain quality evidence leading to analysis creates straightforward and misleading answers to questions presented. Integration is the process of combining information into a Database based on similarities. There are several algorithms, which can be used in data sets based on given condition. [7] However, there is no general consolidation. An algorithm exists, so it is important to choose appropriate methods of integration. Customer segmentation used to be done manually and wasn't too precise. They would manually create and populating different data tables, and analyze the data.

III.EXISTING SYSTEM

In the existing system only KNN algorithm is used for the project which cannot define all the essential factors or features for efficient precision. Recently, Big Data research has gained momentum. Where big data describes a large number of formal and informal data, which cannot be analysed using traditional methods and algorithms. No proper data visualization is used in the existing system such that only tech geek can understand the results from the implementation.

IV.PROBLEM DEFINATION

The marketing methods able to get results by using the traditional methods by the assumption that every customer buy whatever they need without going to the other shops and such that only such type of customers are attracted to the business and the products availability will be limited. For instance if we make groups and assign customers into those categories such that where one size fits method. This can be obtained through only the successful segmentation. The segmented customers are grouped into clusters where all of them almost likes the same things and think similarly, This helps to drive dynamic content and personalization tactics for increase in the business. However, in order for the division to be used effectively, it needs to take into account that different customers are buying for different reasons, and retailers need to use wisely a lot of considerations that could affect their buying decisions. A Harvard Business School professor even said that of the 30,000 new products introduced each year, 95% fail because of poor market performance. There are numerous reasons why the company fail in selecting their target group. It is a wild guess that segmentation is only done on the one factor like demographics or the cluster is too broad to be under the one roof. There may be no strategic goal, such as lead discovery or customer retention. Or, it may be that a lack of One Customer View and aggregated data has led to a misunderstanding or misinterpretation of your customer base. So for segmentation in this project we use almost all factors like demographic, geographic, psychographic and Behavioural.

V.PROPOSED SYSTEM

In Proposed system we Classify customers into segments and Anticipate the purchases that will be made by a customer, during the following year.For classification, I took data from Kaggle. After pre processing the data we will visualize the data and its factors in a graph model using matploit and seaborn libraries. To handle the big data problem I used all the classification algorithms and selected best classifiers in each algorithm and made the prediction by combining all the parameters

For better understanding even for the non tech geek I used word cloud and made the visualizations of each group as samples.In this project large data set is used such that the results which are produced are not biased and the system which is proposed can be used for real time purpose. Ensembling the algorithms and producing the results make the prediction more accurate. The geographical parameters are also took into consideration before classification..

VI. SYSTEM DESIGN

Proposed Model for Customer Grouping: The proposed model helps us in grouping only after providing the suitable data for retrieving the information. In this model the company provides the data to the model and then perform the ensemble model on the data for segregating the customers for increasing their sales. The database used here for machine learning comes from sales of the UK retailer business. The goal of segregation is to increase the sales of the company and attract new customers by showing the their needs in the websites. The data set consists of above 5lakh rows and 8 columns.

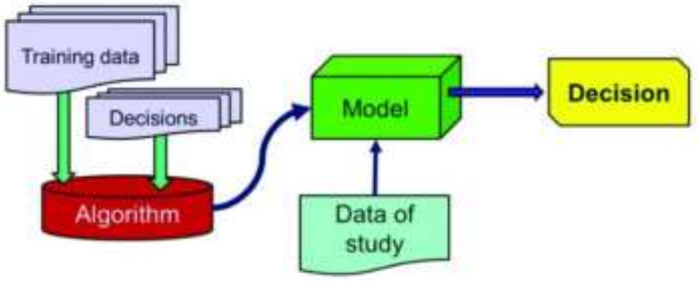


Figure: Training and Decision process of ML Model.

VII. METHODOLOGY

Classification Algorithm

The attributes mentioned in Table 1 are presented as inputs to various ML algorithms such as Random Forest, Decision Tree, Logistic Retreat, Naive Bayes, XG Boost and Light GBM split strategies. The set of input data is divided into 80% of the training database and the remaining 20% of the test database. Database training is a database used to train a model. The test database is used to test the performance of a trained model. In each algorithms the performance is calculated and analyzed based on the various metrics used such as accuracy, precision, recall and F scoring scores as described further. The various algorithms tested in this paper are listed below.

The ML process starts in the pre-processing phase of data followed by feature selection and feature rating and we will use it with the following algorithms such as KNN, SVM, Decision trees, XGBOOST , Support vector Machine and Random Forest algorithms.

Logistic Regression : Regression usually refers to the analysis of segmentation problems, despite your regression name in the segmentation algorithm. It can be used for both binary classification and multiple classification. The predictable rate in depreciation is by category. The sigmoidal function is used to reduce the value from a range of 0 to 1.

XGBOOST: The gradient boosted trees technique is implemented in XGBOOST, a popular and efficient open-source implementation. Gradient boosting is a supervised learning approach that combines the estimates of a collection of smaller, weaker models to attempt to accurately predict a target variable. The data is built in a sequential manner, with each successive data aiming to minimise the entire data's faults. Extreme Gradient Boosting is abbreviated as XGBoost. The word xgboost, on the other hand, alludes to the technical objective of pushing the computational resources for boosted tree algorithms to their limits. XGBoost is a software library that you may download and install on your computer, then use from a number of different interfaces.

Support vector machines: SVM selects extreme points / vectors that help create hyperplane. These extreme cases are called supporting vectors, which is why the algorithm is called Vector Support Machine. Consider the diagram below in which there are two distinct categories divided by resolution or hyperplane.

Random Forest: Random forest are built from decision trees. Generally, tree works well with the data they have or familiar but fails to classify new samples. Random forest adds up the simplicity of the decision tree with flexibility resulting in the large improvement and in the accuracy. At each time we take the bootstrapped data and consider only a subset of the variables at each step which results in a wide variety of trees by applying the decision tree approach. The final result depends on the forests (multiple trees) that we created with the variety of trees. The remaining data after bootstrapping at each step is called out-of-bag data. The correct classification of the out-of-bag data is solemnly responsible for the accuracy of the random forest algorithm. It is an example of the bagging technique.

KNN: K-means the algorithm is one of the most popular segmentation algorithms. This integration algorithm depends on it centro, where each data point is placed in one of scattered, pre-filtered in the K-algorithm. Creating collections corresponding to hidden patterns in data that provides the information needed to help determine execution. process.

Decision Tree: In Decision tree algorithm the data is split by using the conditions and create the classes for each branch. These trees can continuously take the real number values it can also called as regression trees. In this algorithm the identification of the root node parameter is difficult. The process of identifying is called attribute selection. For this we need to take 2 factors under consideration they are information gain and Gini index.

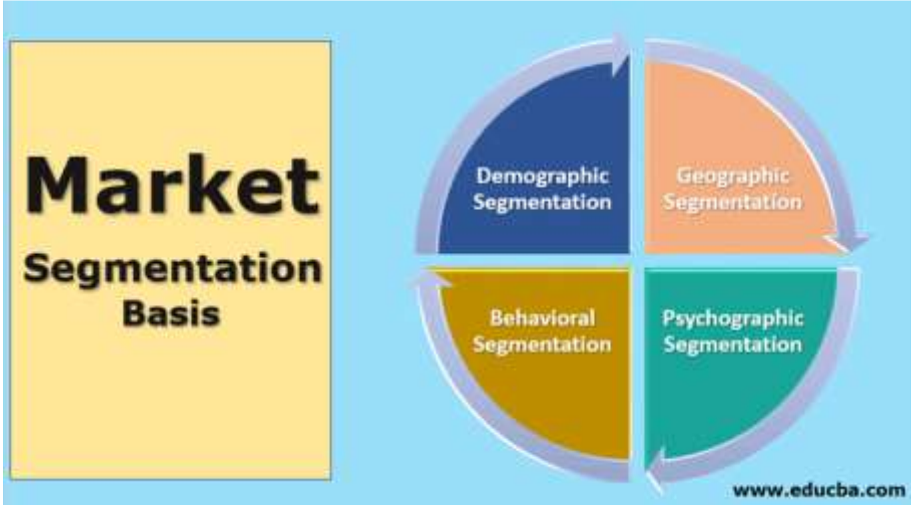
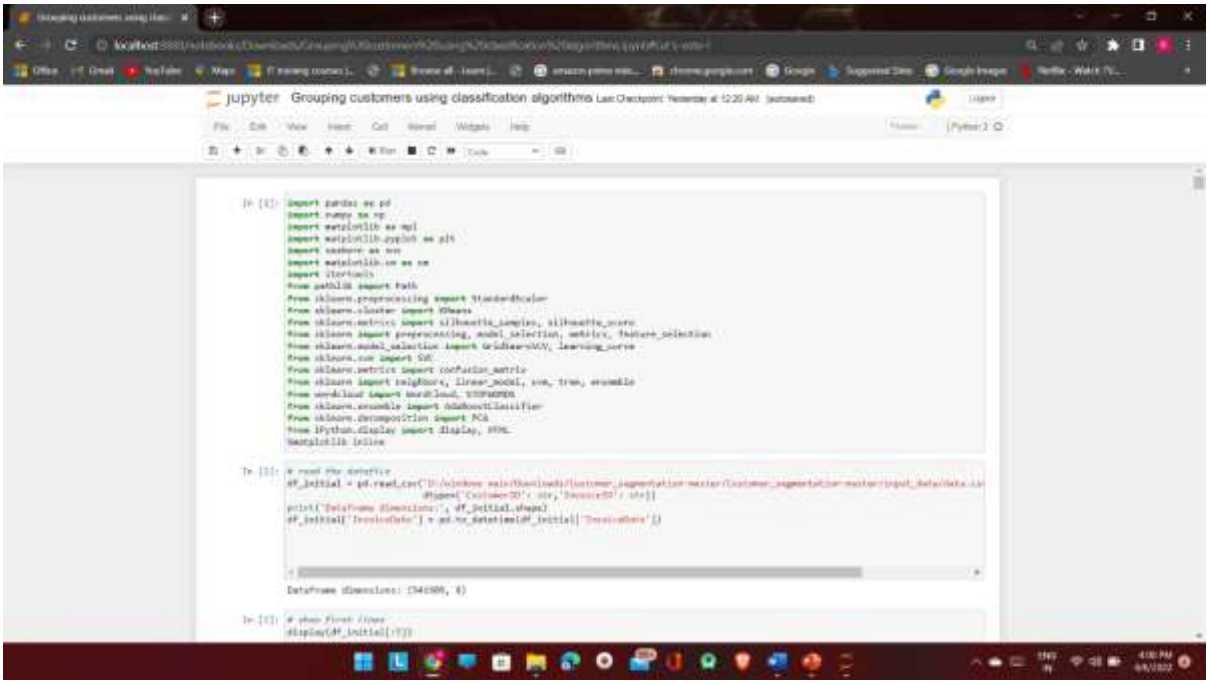


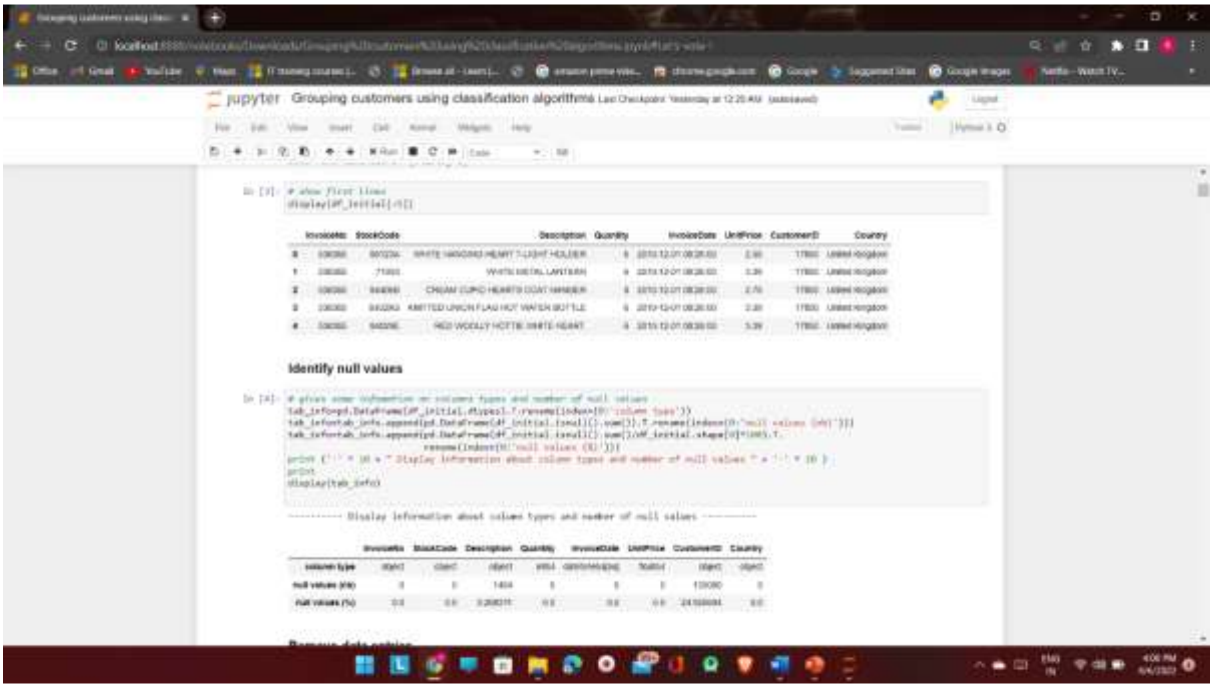
Figure : Market Grouping Factors.



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Python 3

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib as mpl
        import matplotlib.pyplot as plt
        import seaborn as sns
        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
        %matplotlib inline

In [2]: # read the datafile
        df_initial = pd.read_csv('D:/windows main/Downloads/customer_segmentation-master/Customer_segmentation-master/input_data/data.cs
                              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])
```

Figure 8.1: Importing the essential dependencies



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Python 3

```
In [3]: # show first lines
        display(df_initial[:5])
```

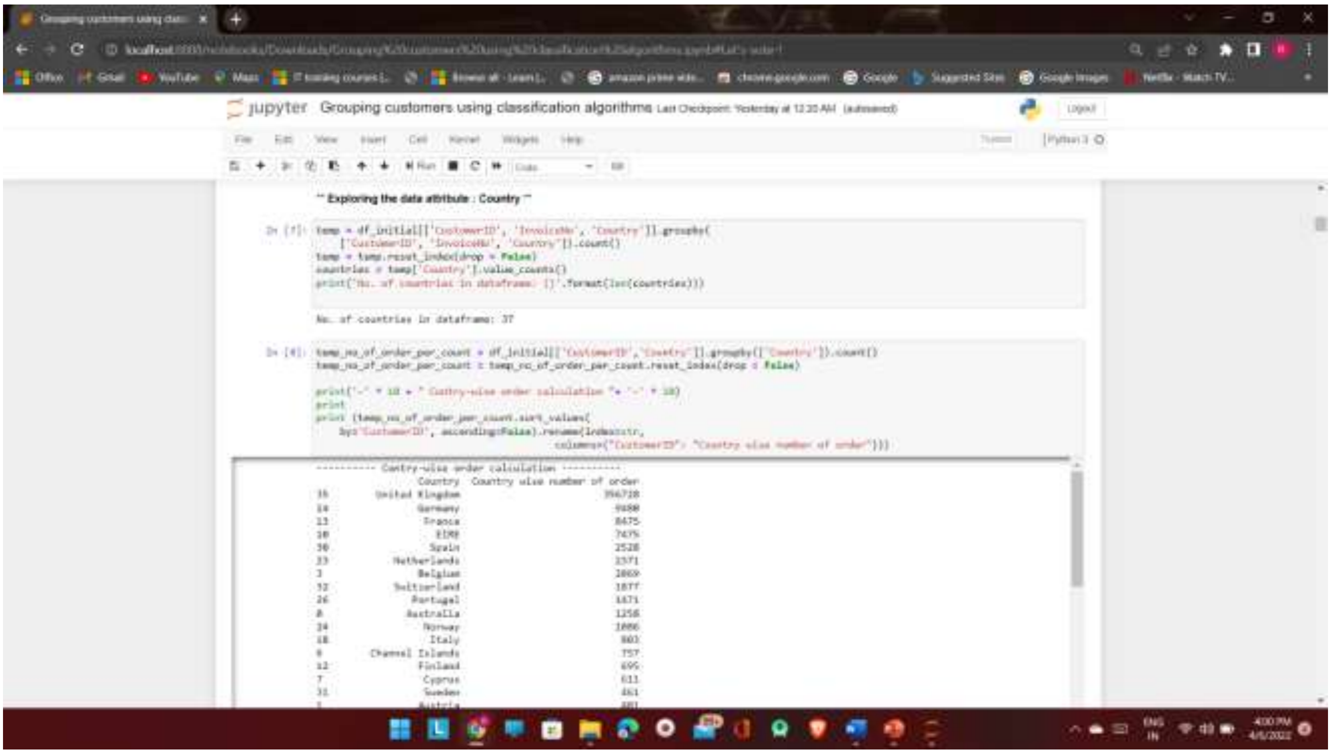
	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT 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

Identify null values

```
In [4]: # gives some infos 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 values (nb)'}))
        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 " + '-' * 10 )
        print
        display(tab_info)

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

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
column type	object	object	object	int64	datetime64[ns]	float64	object	object
null values (nb)	0	0	1454	0	0	0	135080	0
null values (%)	0.0	0.0	0.268311	0.0	0.0	0.0	24.926694	0.0

Figure 8.2: Data preprocessing and data cleaning


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

''' Exploring the data attribute : Country '''

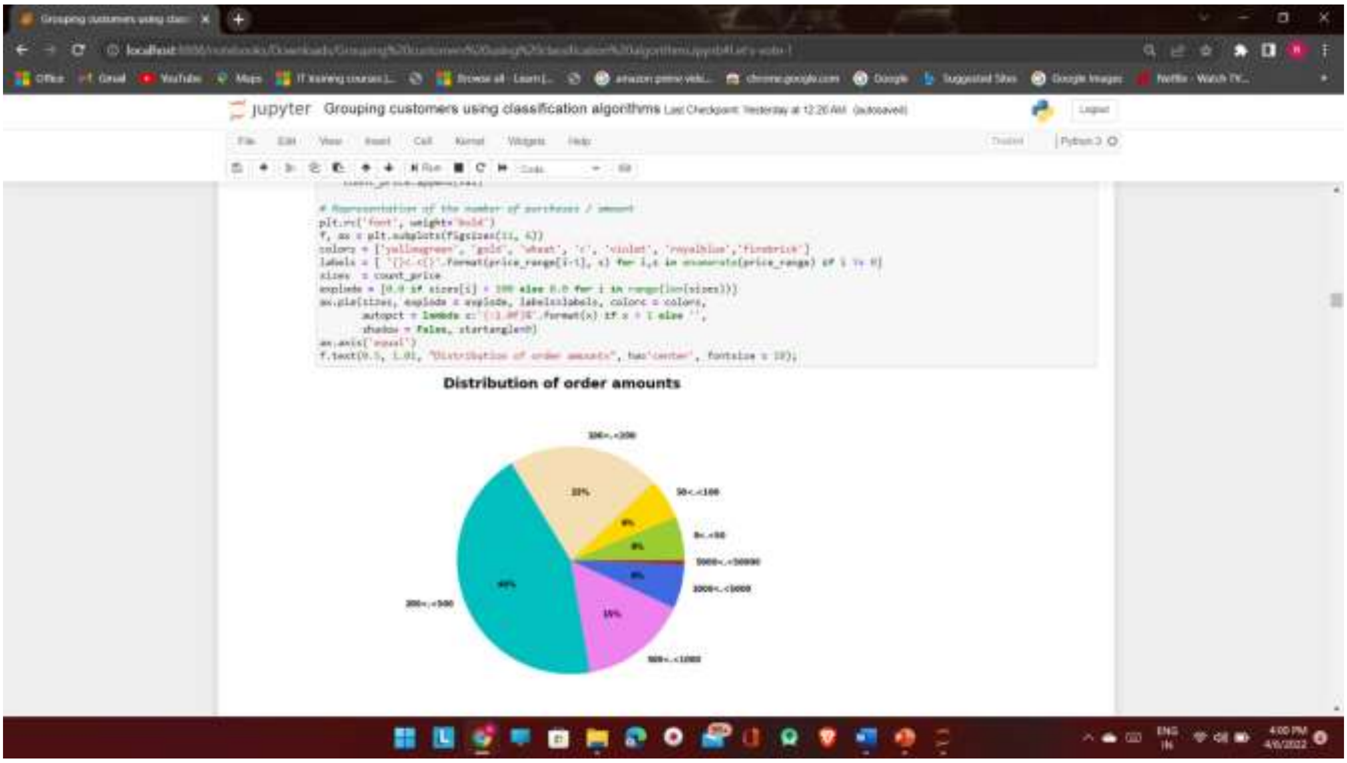
```
In [7]: temp = df_initial[['CustomerID', 'InvoiceNo', 'Country']].groupby(
            ['CustomerID', 'InvoiceNo', 'Country']).count()
        temp = temp.reset_index(drop = False)
        countries = temp['Country'].value_counts()
        print('No. of countries in dataframe: {}'.format(len(countries)))
```

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

```
In [8]: temp_no_of_order_per_count = df_initial[['CustomerID','Country']].groupby(['Country']).count()
        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 order"}))
```

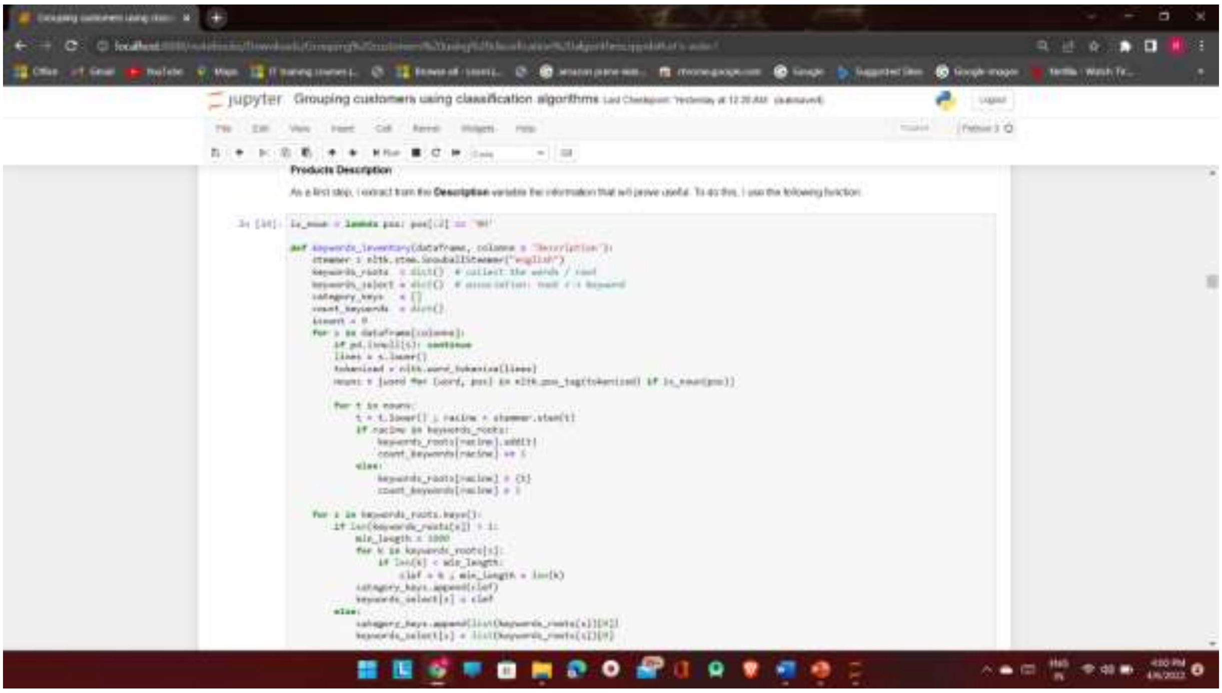
```
---------- Contry-wise order calculation ----------
           Country  Country wise number of order
35  United Kingdom                        356728
14         Germany                          9480
13          France                          8475
10            EIRE                          7475
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
```

Figure 8.3: Exploring the data and geographical purchases


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:26 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Distribution of order amounts
100<.<200
50<.<100
0<.<50
5000<.<50000
1000<.<5000
500<.<1000
200<.<500
22%
6%
6%
6%
40%
15%

```
# 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 != 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, "Distribution of order amounts", ha='center', fontsize = 18);
```

Figure 8.4: Flow chart of the transactions and order of amounts.

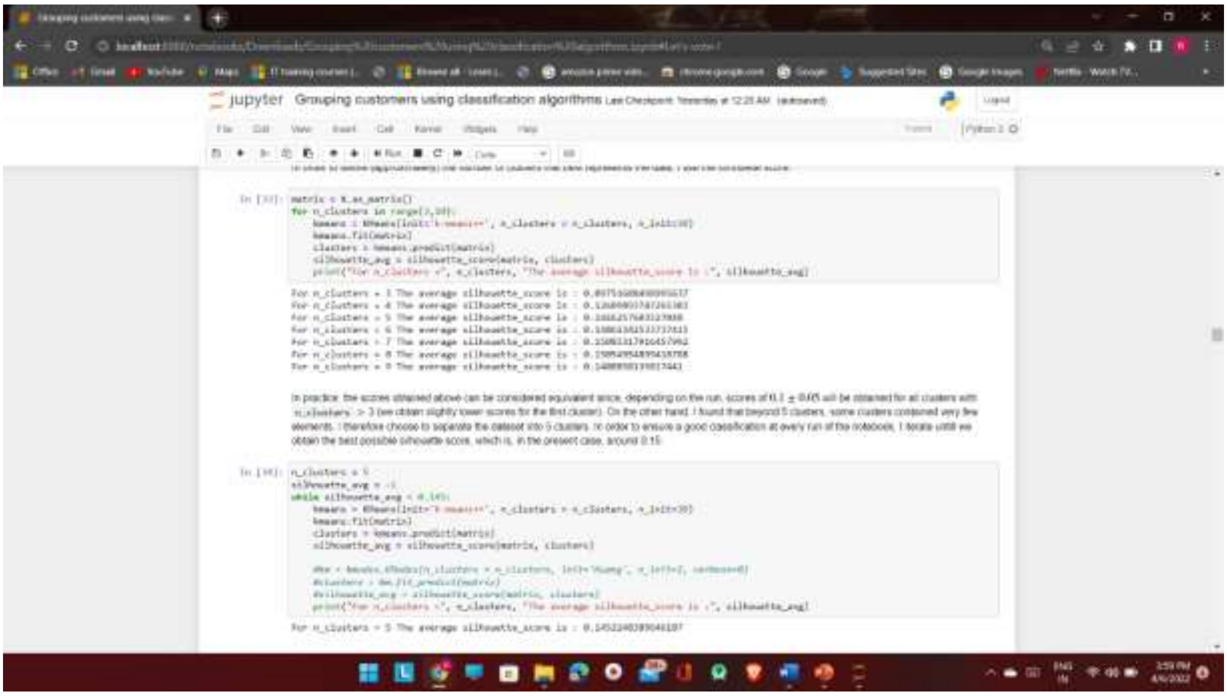

jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:30 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3

Products Description

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

```
In [16]: 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]
```

Figure 8.5: Product Description and clusters are created by these keywords


jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:28 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Python 3

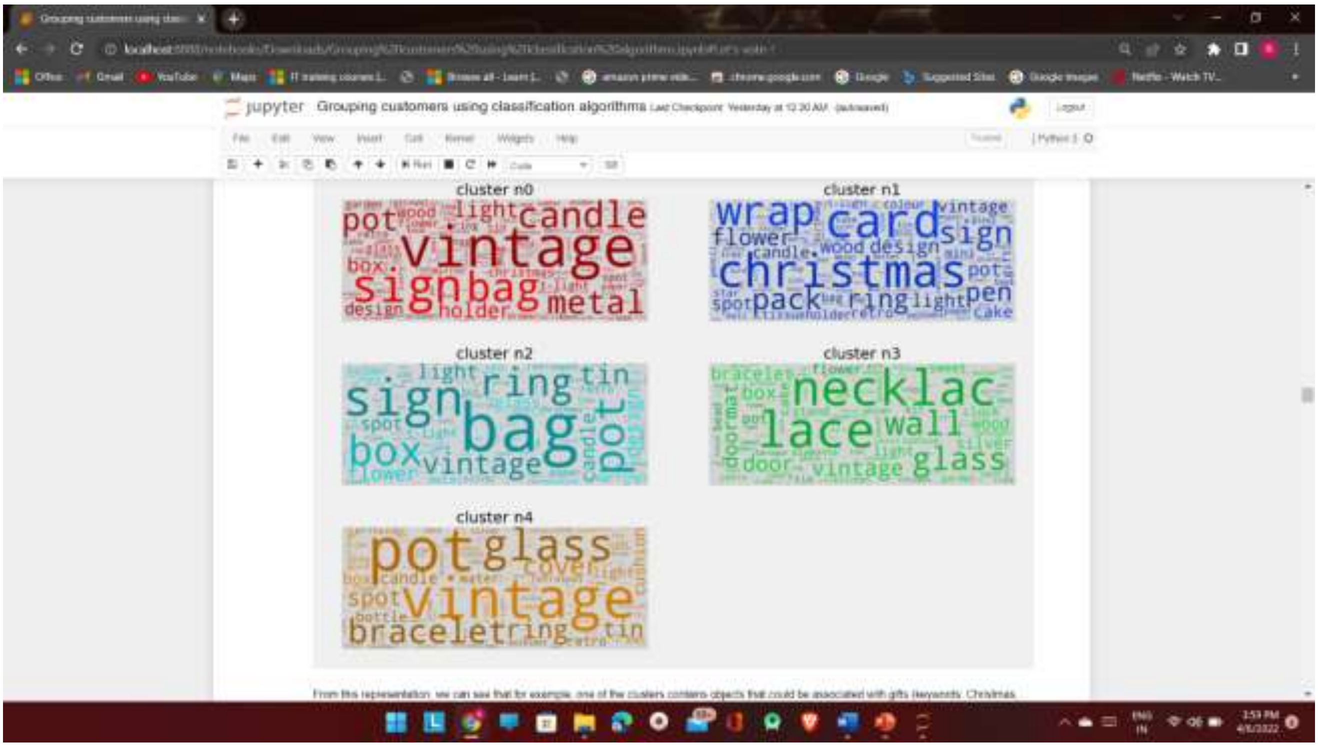
In order to define (approximately) the number of clusters that best represents the data, I use the silhouette score:

```
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_avg)

         For n_clusters = 3 The average silhouette_score is : 0.[illegible]
         For n_clusters = 4 The average silhouette_score is : 0.[illegible]
         For n_clusters = 5 The average silhouette_score is : 0.[illegible]
         For n_clusters = 6 The average silhouette_score is : 0.[illegible]
         For n_clusters = 7 The average silhouette_score is : 0.[illegible]
         For n_clusters = 8 The average silhouette_score is : 0.[illegible]
         For n_clusters = 9 The average silhouette_score is : 0.[illegible]
```

In practice, the scores obtained above can be considered equivalent since, depending on the run, scores of 0.1 ± 0.05 will be obtained for all clusters with `n_clusters` > 3 (we obtain slightly lower scores for the first cluster). On the other hand, I found that beyond 5 clusters, some clusters contained very few elements. I therefore choose to separate the dataset into 5 clusters. In order to ensure a good classification at every run of the notebook, I iterate until we obtain the best possible silhouette score, which is, in the present case, around 0.15.

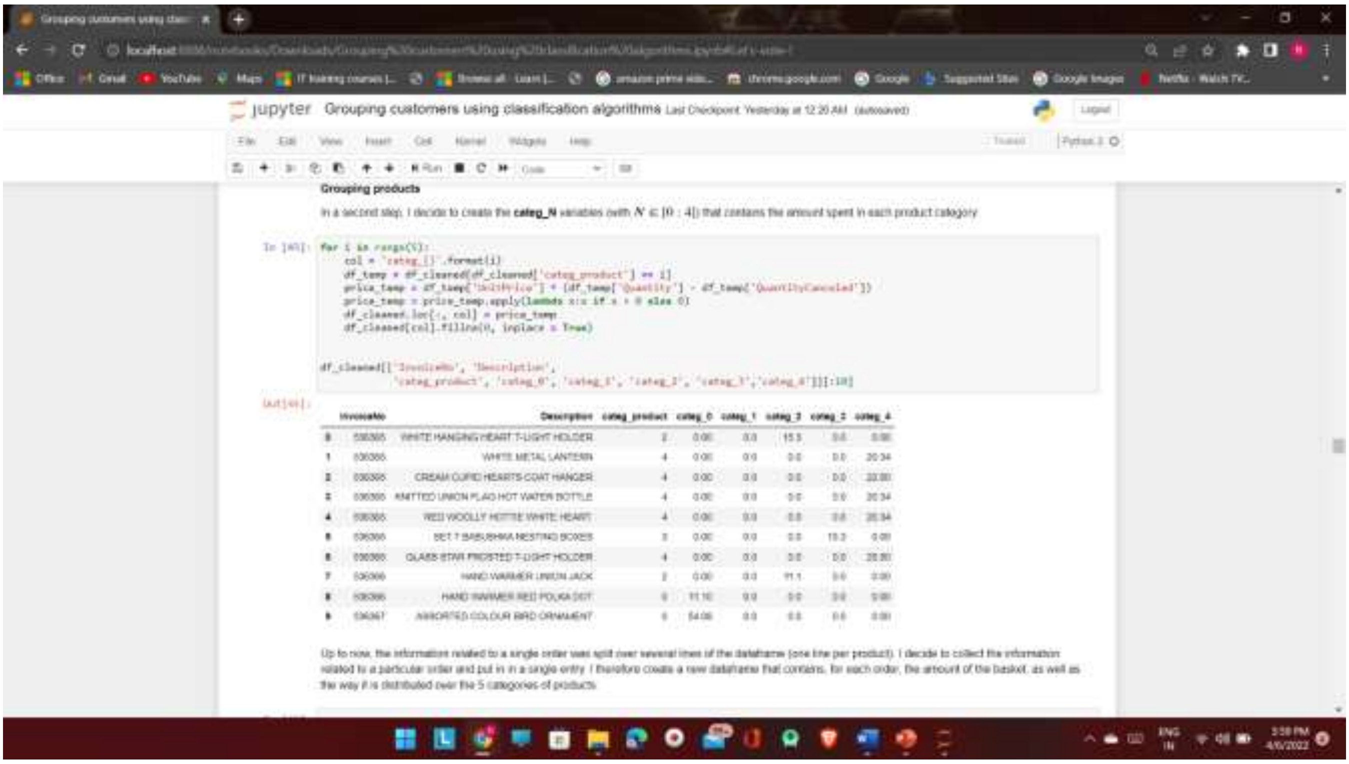
```
In [34]: n_clusters = 5
         silhouette_avg = -1
         while silhouette_avg < 0.145:
             kmeans = KMeans(init='k-means++', n_clusters = n_clusters, n_init=30)
             kmeans.fit(matrix)
             clusters = kmeans.predict(matrix)
             silhouette_avg = silhouette_score(matrix, 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_avg)

         For n_clusters = 5 The average silhouette_score is : 0.145[illegible]
```

Figure 8.6: Silhouette score of the cluster groups are determined.


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
cluster n0
cluster n1
cluster n2
cluster n3
cluster n4

From this representation, we can see that for example, one of the clusters contains objects that could be associated with gifts (keywords: Christmas,

Figure 8.7: All the customers are classified by words and visualized


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3

Grouping products

In a second step, I decide to create the **categ_N** variables (with $N \in [0:4]$) that contains the amount spent in each product category

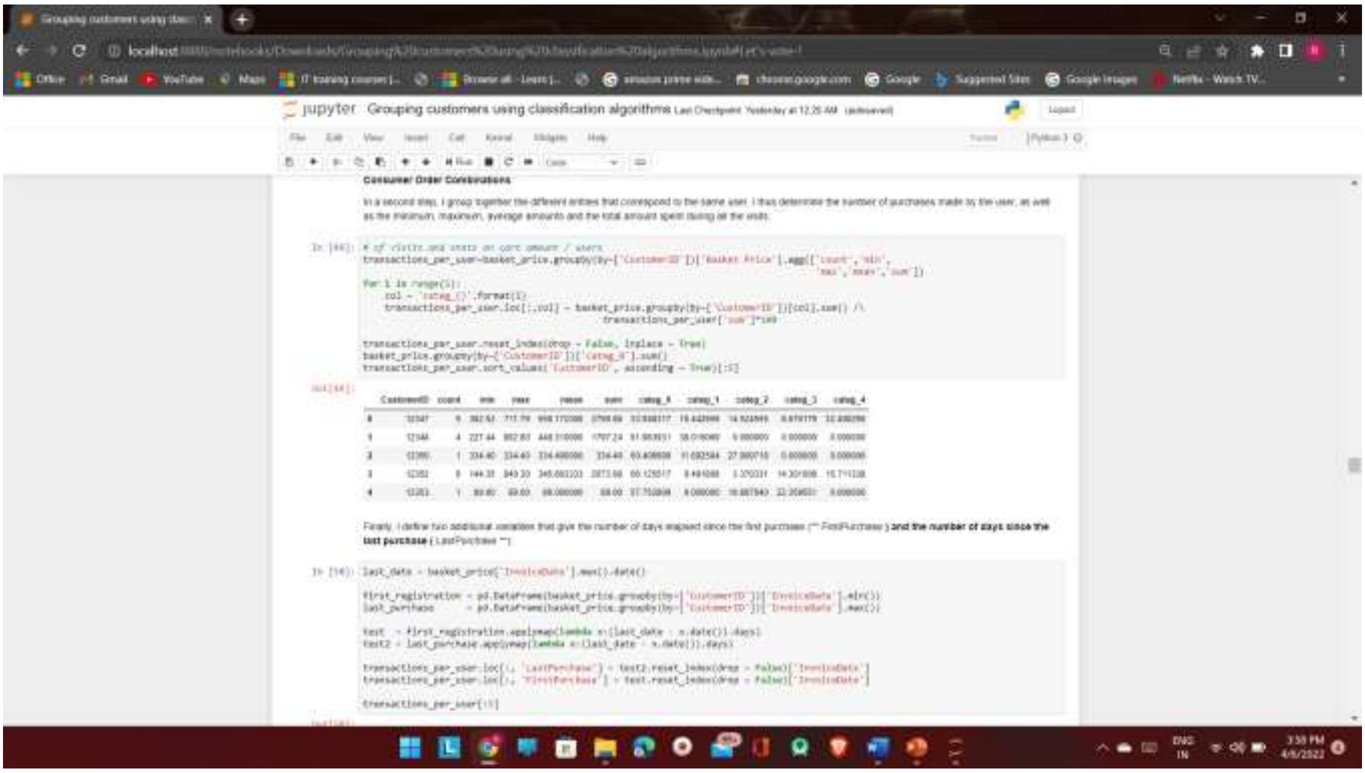
```
In [45]: for i in range(5):
             col = 'categ_{}'.format(i)
             df_temp = df_cleaned[df_cleaned['categ_product'] == i]
             price_temp = df_temp['UnitPrice'] * (df_temp['Quantity'] - df_temp['QuantityCanceled'])
             price_temp = price_temp.apply(lambda x:x if x > 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

Up to now, the information related to a single order was split over several lines of the dataframe (one line per product). I decide to collect the information related to a particular order and put in in a single entry. I therefore create a new dataframe that contains, for each order, the amount of the basket, as well as the way it is distributed over the 5 categories of products.

Figure 8.8: After the customers are grouped now we should group the prodcuts



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:26 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Consumer Order Combinations

In a second step, I group together the different entries that correspond to the same user. I thus determine the number of purchases made by the user, as well as the minimum, maximum, average amounts and the total amount spent during all the visits:

```
In [48]: # of visits and stats on cart amount / users
         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]
```

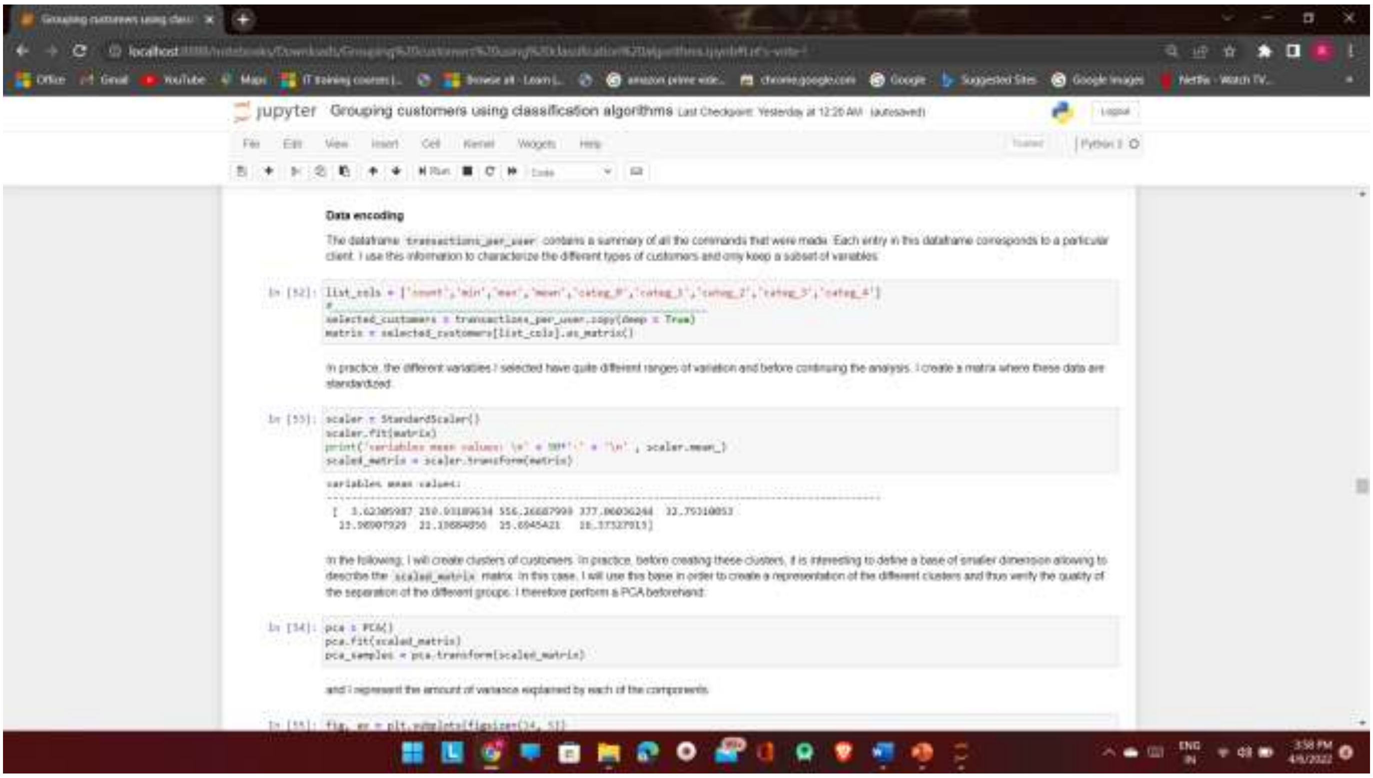
Out[48]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	categ_3	categ_4
0	12347	5	382.52	711.79	558.172000	2790.86	10.938117	19.442099	14.524555	8.670175	32.408290
1	12348	4	227.44	892.80	449.310000	1797.24	41.953951	58.016085	0.000000	0.000000	0.000000
2	12350	1	334.40	334.40	334.400000	334.40	60.406699	11.692584	27.900718	0.000000	0.000000
3	12352	6	144.35	840.30	345.663333	2073.98	66.125017	8.491088	3.370331	14.301006	15.711128
4	12353	1	89.00	89.00	89.000000	89.00	57.752809	0.000000	19.887640	22.359551	0.000000

Finally, I define two additional variables that give the number of days elapsed since the first purchase (" FirstPurchase ") and **the number of days since the last purchase** (LastPurchase "):

```
In [50]: 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]
```

Figure 8.9 : Next I created the combinations of the customers transactions made\



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Data encoding

The dataframe `transactions_per_user` contains a summary of all the commands that were made. Each entry in this dataframe corresponds to a particular client. I use this information to characterize the different types of customers and only keep a subset of variables:

```
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 practice, the different variables I selected have quite different ranges of variation and before continuing the analysis, I create a matrix where these data are standardized:

```
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 250.93189634 556.26687998 377.06036244  32.79310853
  15.90907929  21.19884856  25.6945421   16.37327913]
```

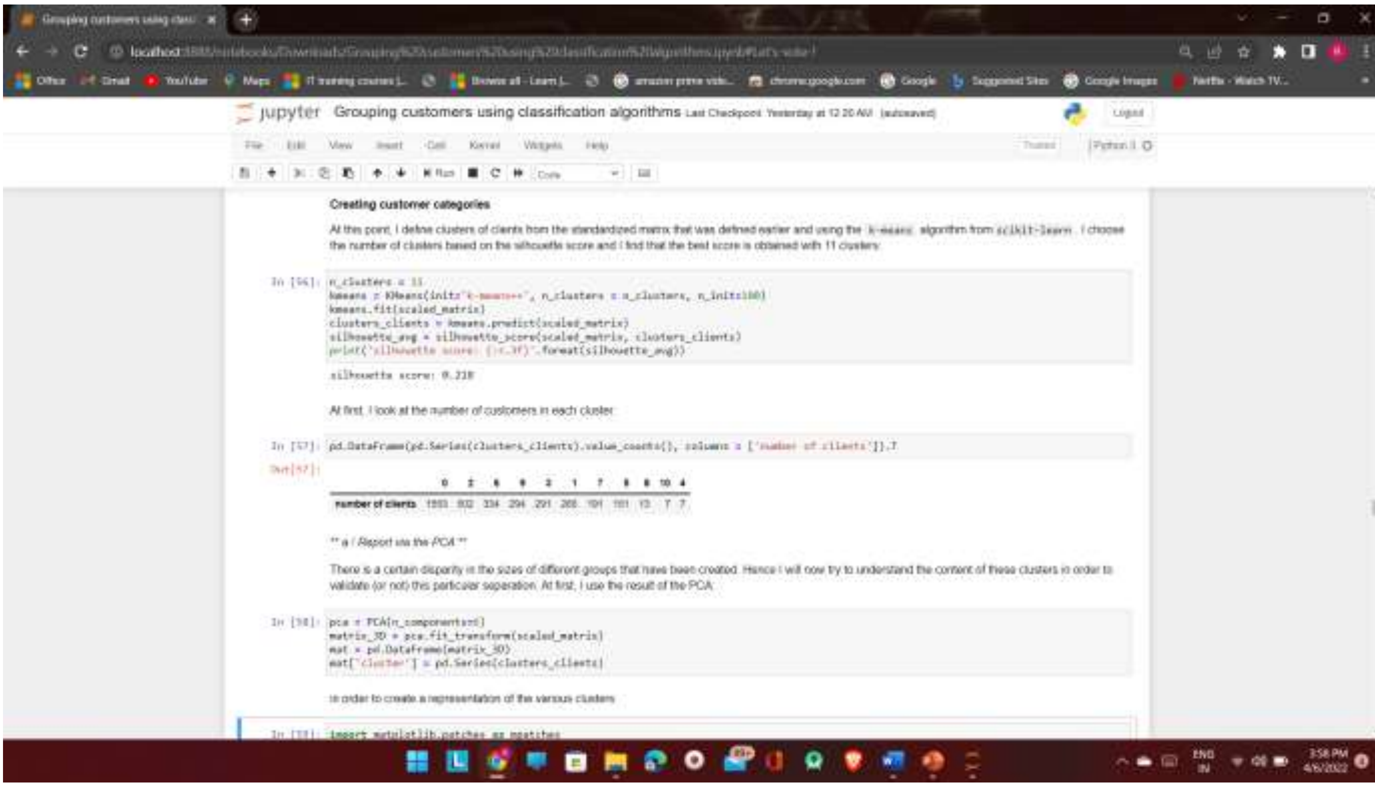
In the following, I will create clusters of customers. In practice, before creating these clusters, it is interesting to define a base of smaller dimension allowing to describe the `scaled_matrix` matrix. In this case, I will use this base in order to create a representation of the different clusters and thus verify the quality of the separation of the different groups. I therefore perform a PCA beforehand:

```
In [54]: pca = PCA()
         pca.fit(scaled_matrix)
         pca_samples = pca.transform(scaled_matrix)
```

and I represent the amount of variance explained by each of the components:

```
In [55]: fig, ax = plt.subplots(figsize=(14, 5))
```

Figure 8.10 : Data encoding turning words into numerical values for processing



jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

Creating customer categories

At this point, I define clusters of clients from the standardized matrix that was defined earlier and using the `k-means` algorithm from `scikit-learn` . I choose the number of clusters based on the silhouette score and I find that the best score is obtained with 11 clusters:

```
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
```

At first, I look at the number of customers in each cluster:

```
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	802	334	294	291	268	191	181	13	7	7

** a / Report via the PCA **

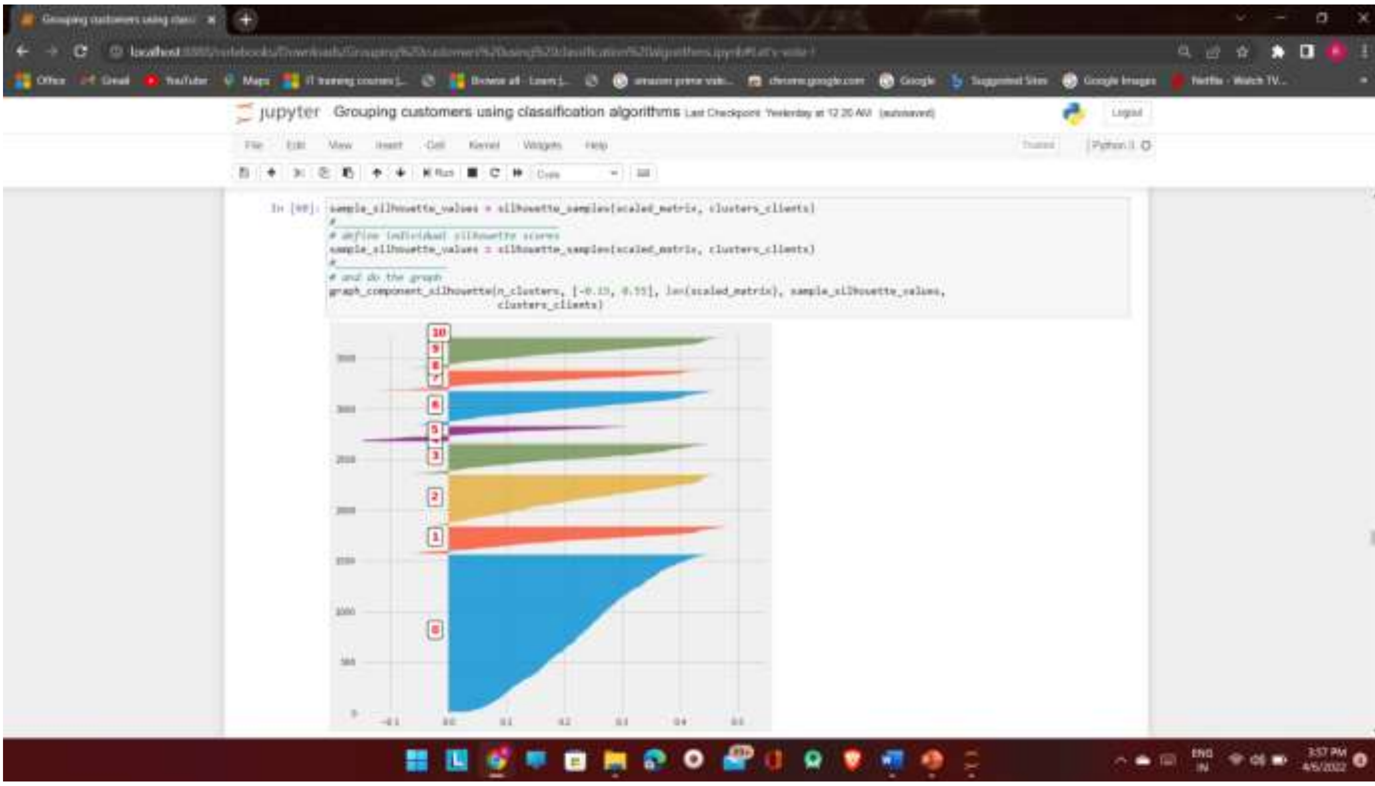
There is a certain disparity in the sizes of different groups that have been created. Hence I will now try to understand the content of these clusters in order to validate (or not) this particular separation. At first, I use the result of the PCA:

```
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 order to create a representation of the various clusters:

```
In [59]: import matplotlib.patches as mpatches
```

Figure 8.11 : Customer categories created.

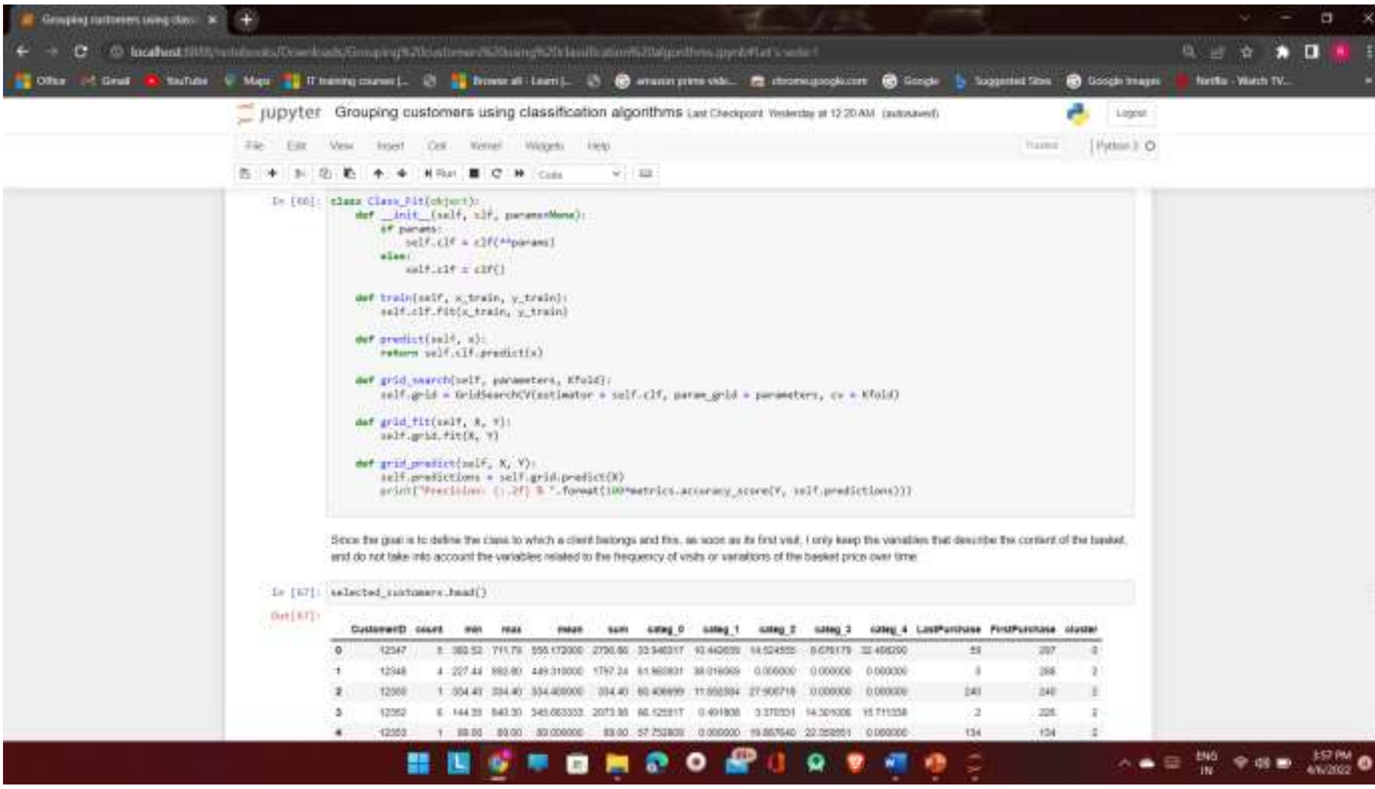


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
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_values,
                                    clusters_clients)
```

10
9
8
7
6
5
4
3
2
1
0
3500
3000
2500
2000
1500
1000
500
0
-0.1
0.0
0.1
0.2
0.3
0.4
0.5

Figure 8.12 : Customer categories are visualized



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Logout

```
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)))
```

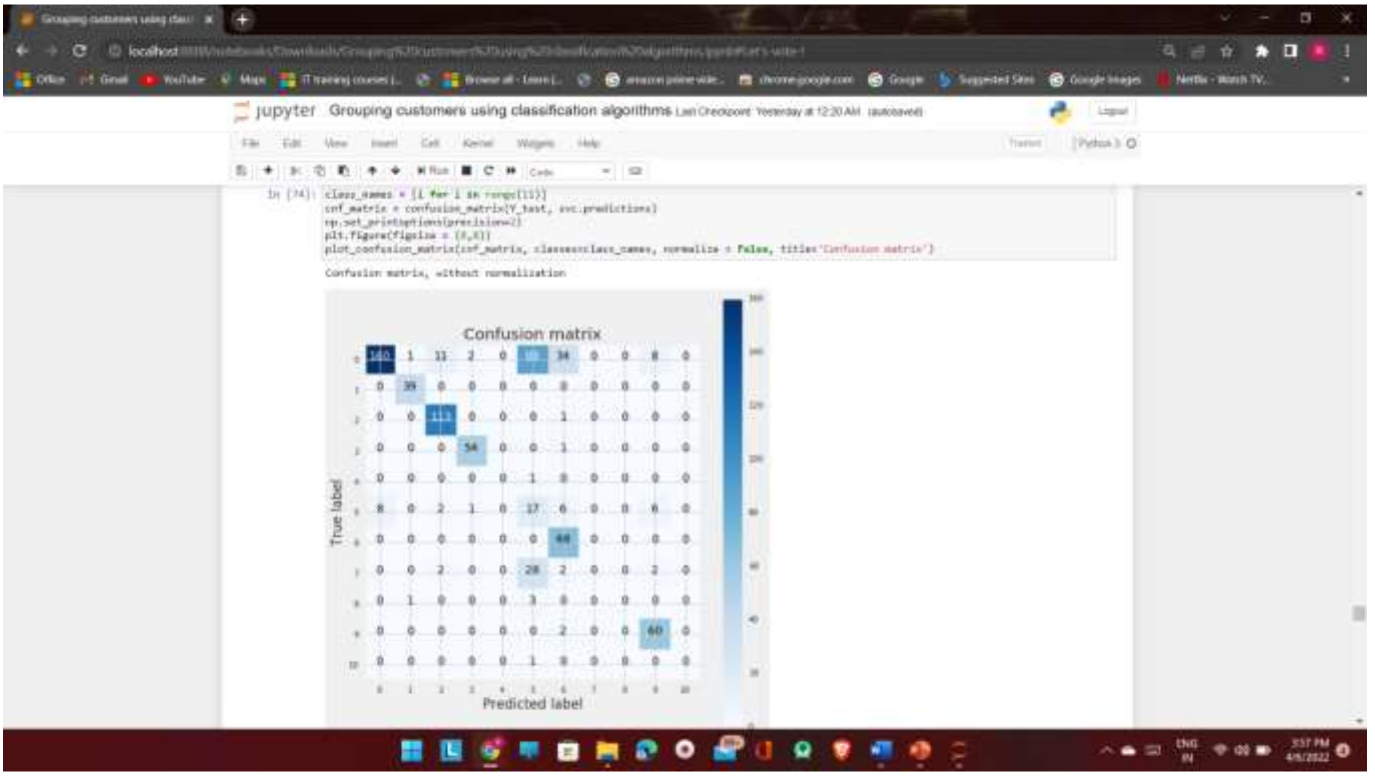
Since the goal is to define the class to which a client belongs and this, as soon as its first visit, I only keep the variables that describe the content of the basket, and do not take into account the variables related to the frequency of visits or variations of the basket price over time

```
In [67]: selected_customers.head()
```

Out[67]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	categ_3	categ_4	LastPurchase	FirstPurchase	cluster
0	12347	5	382.52	711.79	558.172000	2790.86	33.940317	10.442659	14.524555	8.675179	32.408290	59	297	0
1	12348	4	227.44	892.80	449.310000	1797.24	61.960831	38.039169	0.000000	0.000000	0.000000	5	288	2
2	12350	1	334.40	334.40	334.400000	334.40	60.406699	11.692584	27.900718	0.000000	0.000000	240	240	2
3	12352	6	144.35	840.30	345.663333	2073.98	66.125517	0.491808	3.370331	14.301006	15.711338	2	226	2
4	12353	1	89.00	89.00	89.000000	89.00	57.752809	0.000000	19.887640	22.359551	0.000000	134	134	2

Figure 8.13 : Program for creating the categories display and enhancement

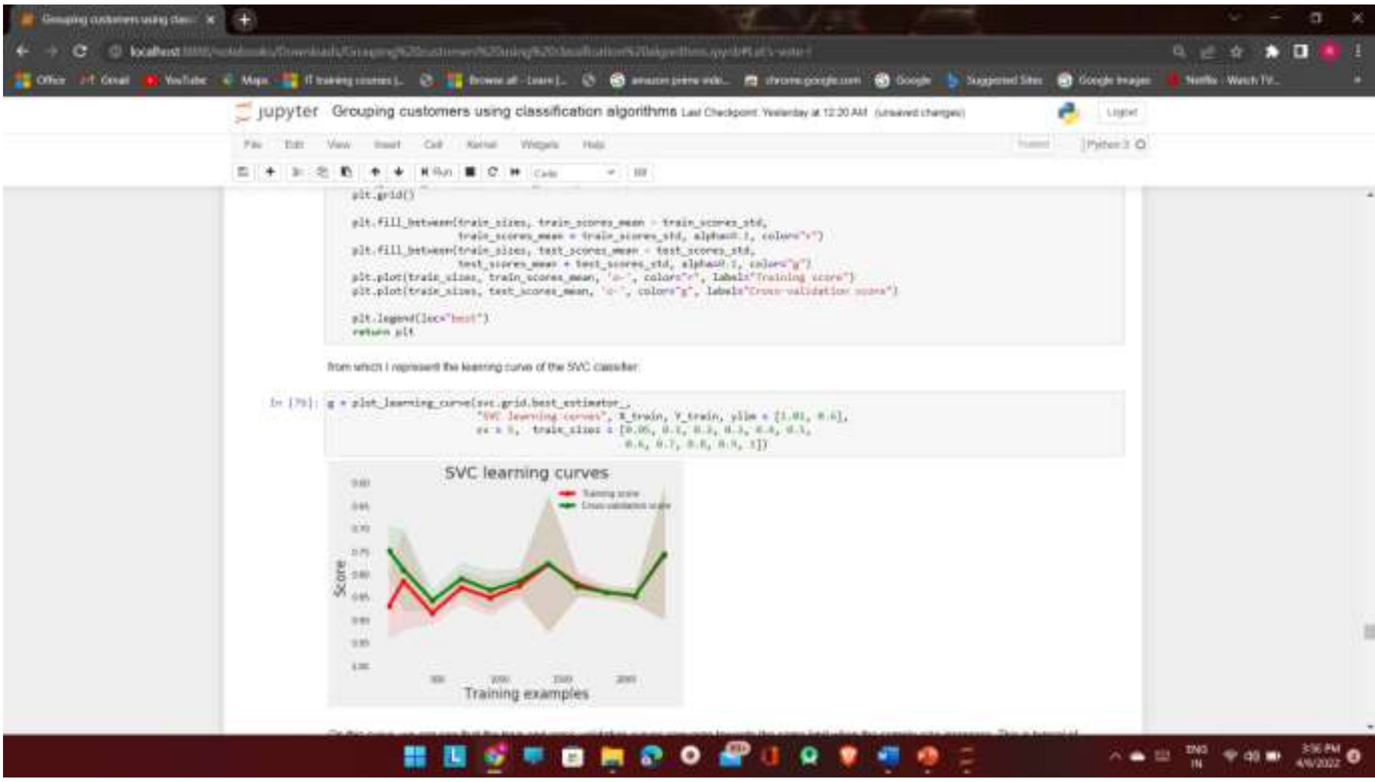


jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Logout
Confusion matrix
True label
Predicted label

```
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 matrix')

Confusion matrix, without normalization
```

	0	1	2	3	4	5	6	7	8	9	10
0	140	1	11	2	0	[illegible]	34	0	0	8	0
1	0	39	0	0	0	0	0	0	0	0	0
2	0	0	113	0	0	0	1	0	0	0	0
3	0	0	0	54	0	0	1	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0
5	8	0	2	1	0	17	6	0	0	6	0
6	0	0	0	0	0	0	68	0	0	0	0
7	0	0	2	0	0	28	2	0	0	2	0
8	0	1	0	0	0	3	0	0	0	0	0
9	0	0	0	0	0	0	2	0	0	60	0
10	0	0	0	0	0	1	0	0	0	0	0

Figure 8.14 : Displaying the confuse matrix of the MLmodel



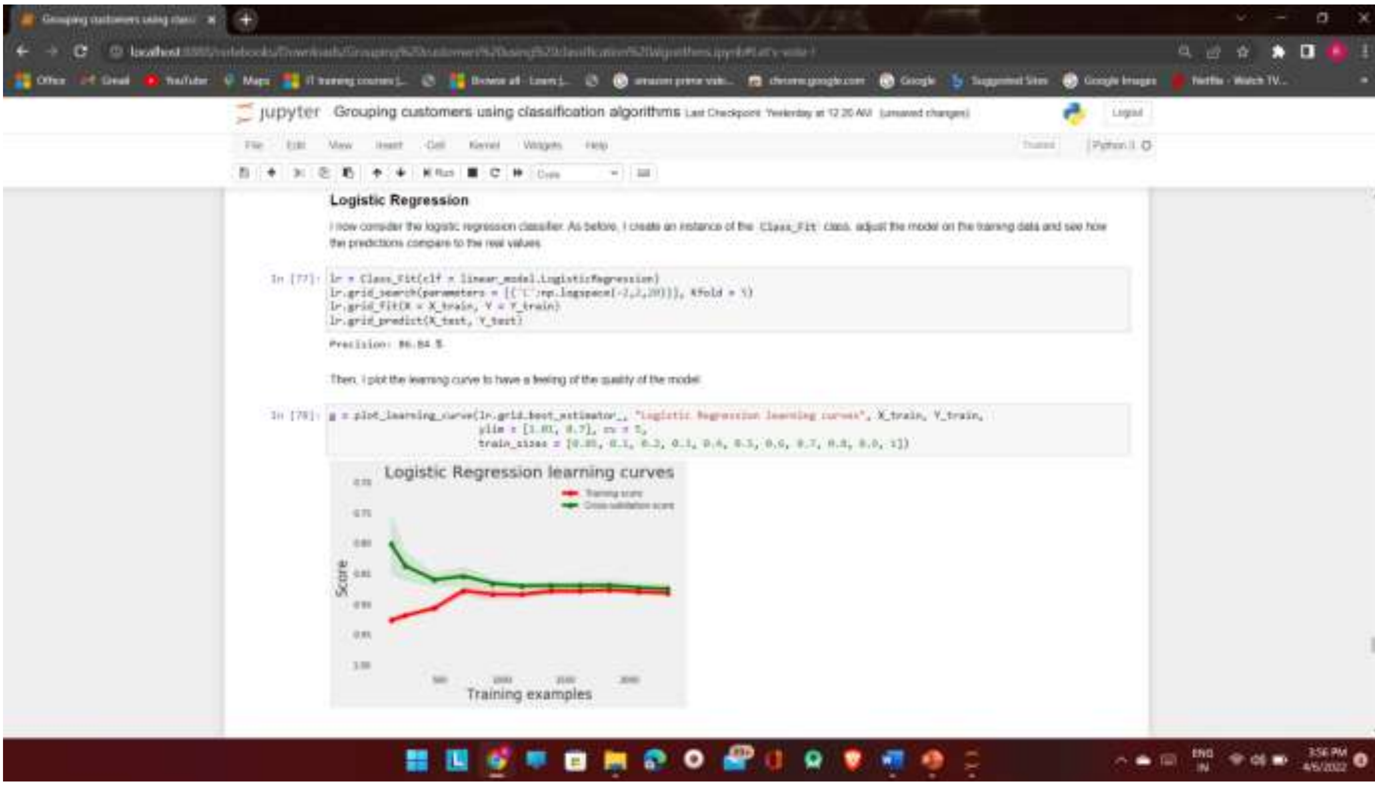
jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

```
    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
```

from which I represent the learning curve of the SVC classifier:

```
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])
```

Figure 8.15 : Accuracy of the SVC.



jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code

Logistic Regression

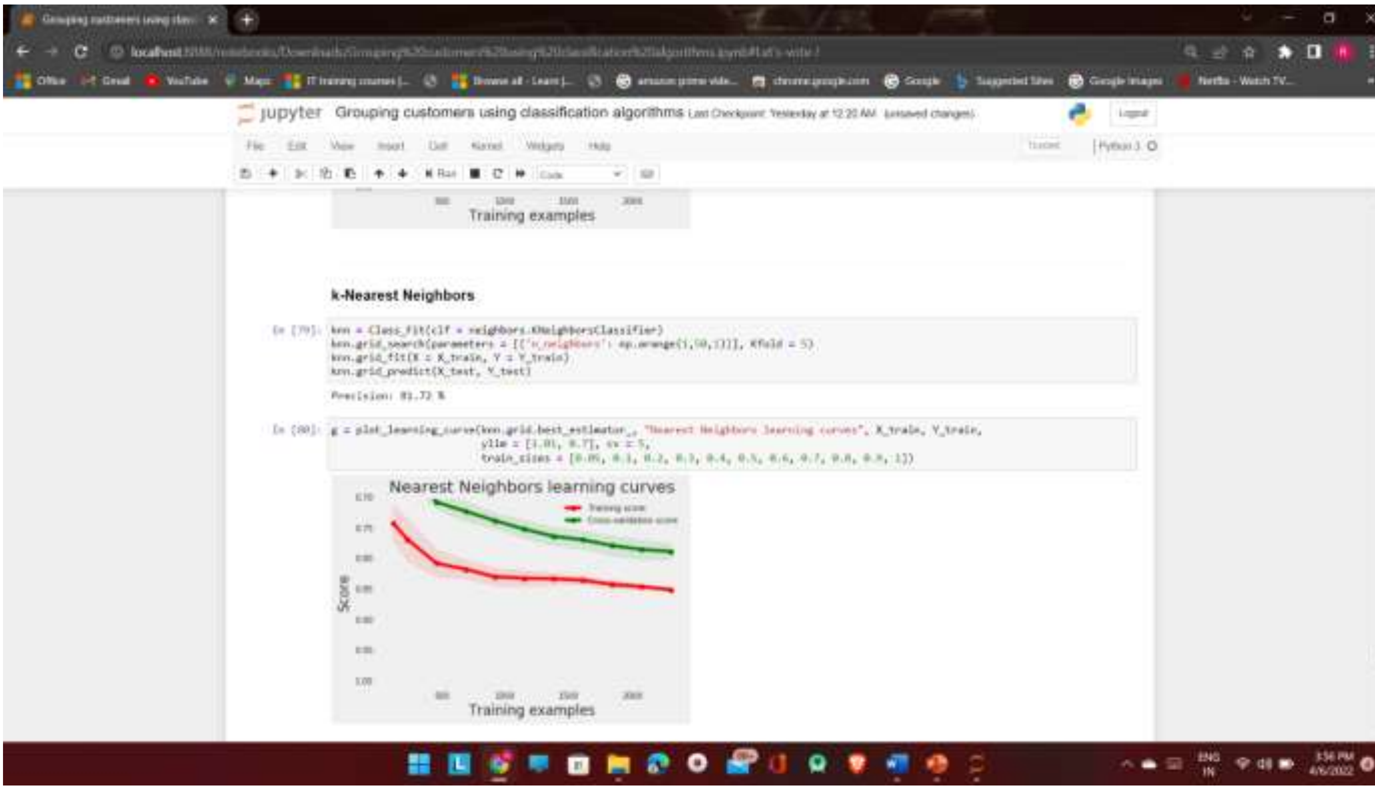
I now consider the logistic regression classifier. As before, I create an instance of the `Class_Fit` class, adjust the model on the training data and see how the predictions compare to the real values

```
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 %
```

Then, I plot the learning curve to have a feeling of the quality of the model

```
In [78]: 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])
```

Figure 8.16 : Accuracy of the Logistic regression

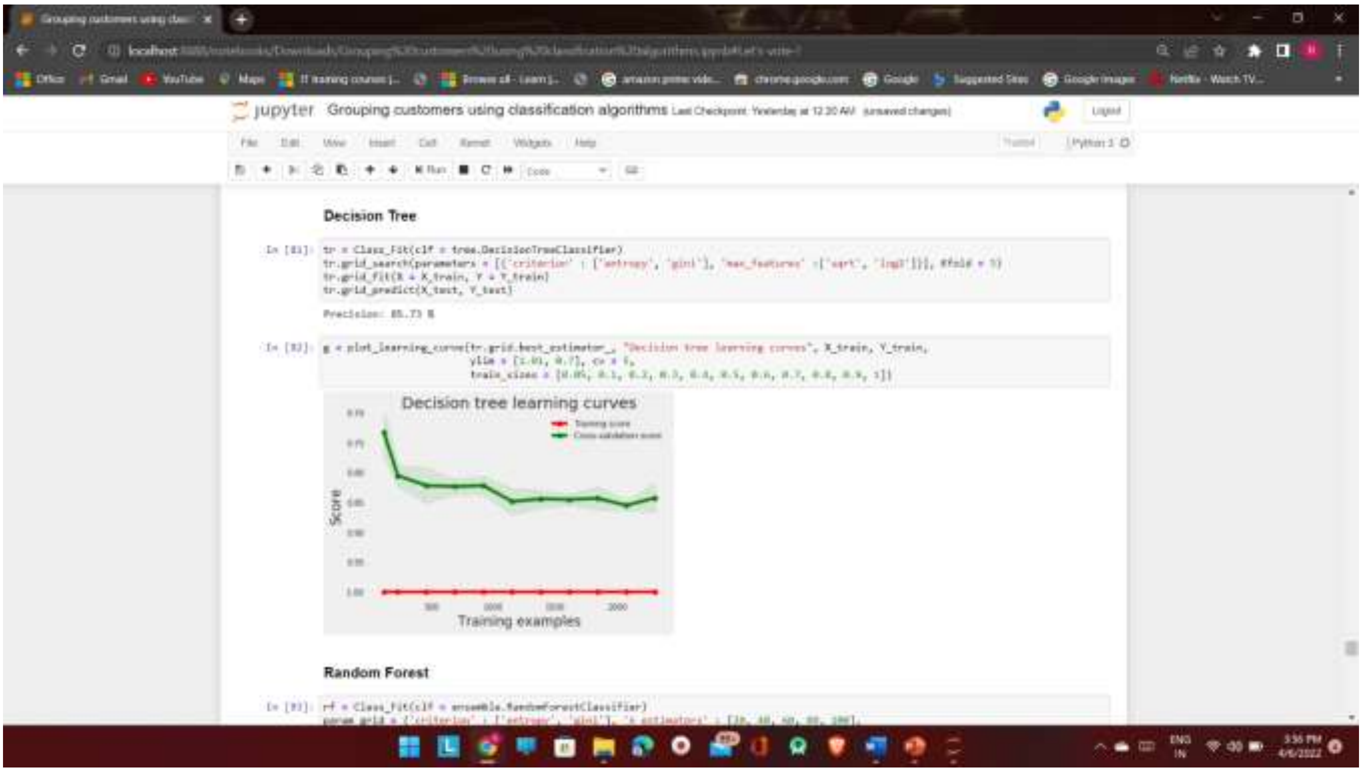


k-Nearest Neighbors

```
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_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])
```

Figure 8.17 : Accuracy of the KNN



Decision Tree

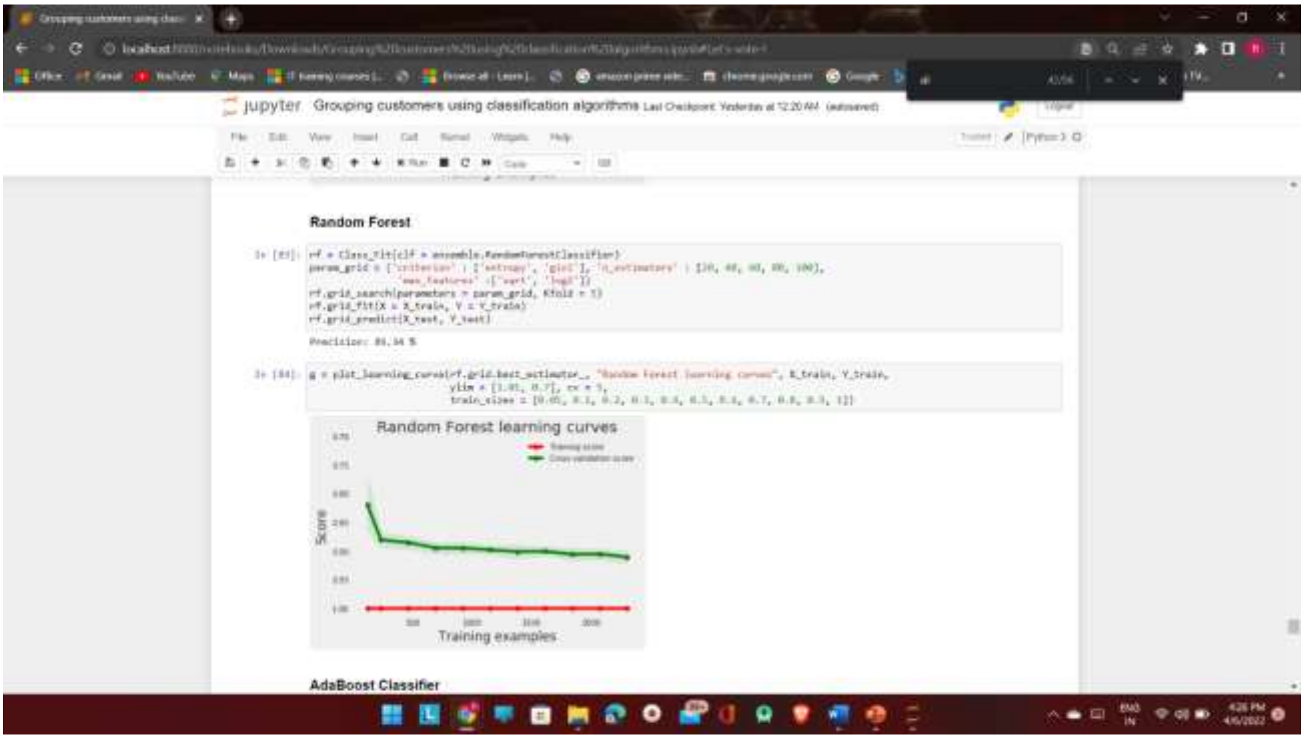
```
In [81]: tr = Class_Fit(clf = tree.DecisionTreeClassifier)
         tr.grid_search(parameters = [{'criterion' : ['entropy', 'gini'], 'max_features' :['sqrt', 'log2']}], Kfold = 5)
         tr.grid_fit(X = X_train, Y = Y_train)
         tr.grid_predict(X_test, Y_test)

Precision: 85.73 %
```

```
In [82]: 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

```
In [83]: rf = Class_Fit(clf = ensemble.RandomForestClassifier)
         param_grid = {'criterion' : ['entropy', 'gini'], 'n_estimators' : [20, 40, 60, 80, 100],
```

Figure 8.18 : Accuracy of the Decision tree.



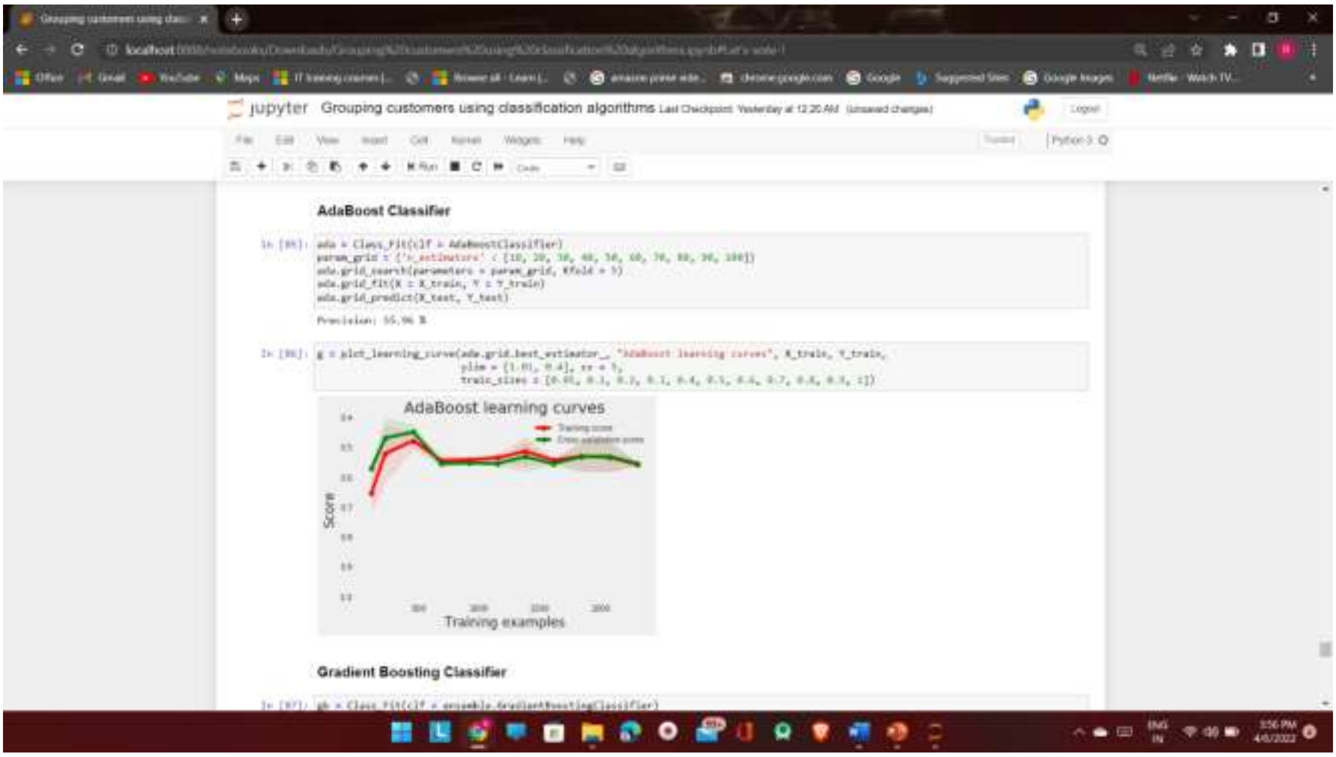
jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (autosaved)

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, 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

Figure 8.19: Accuracy of the Random Forest



jupyter Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (unsaved changes)

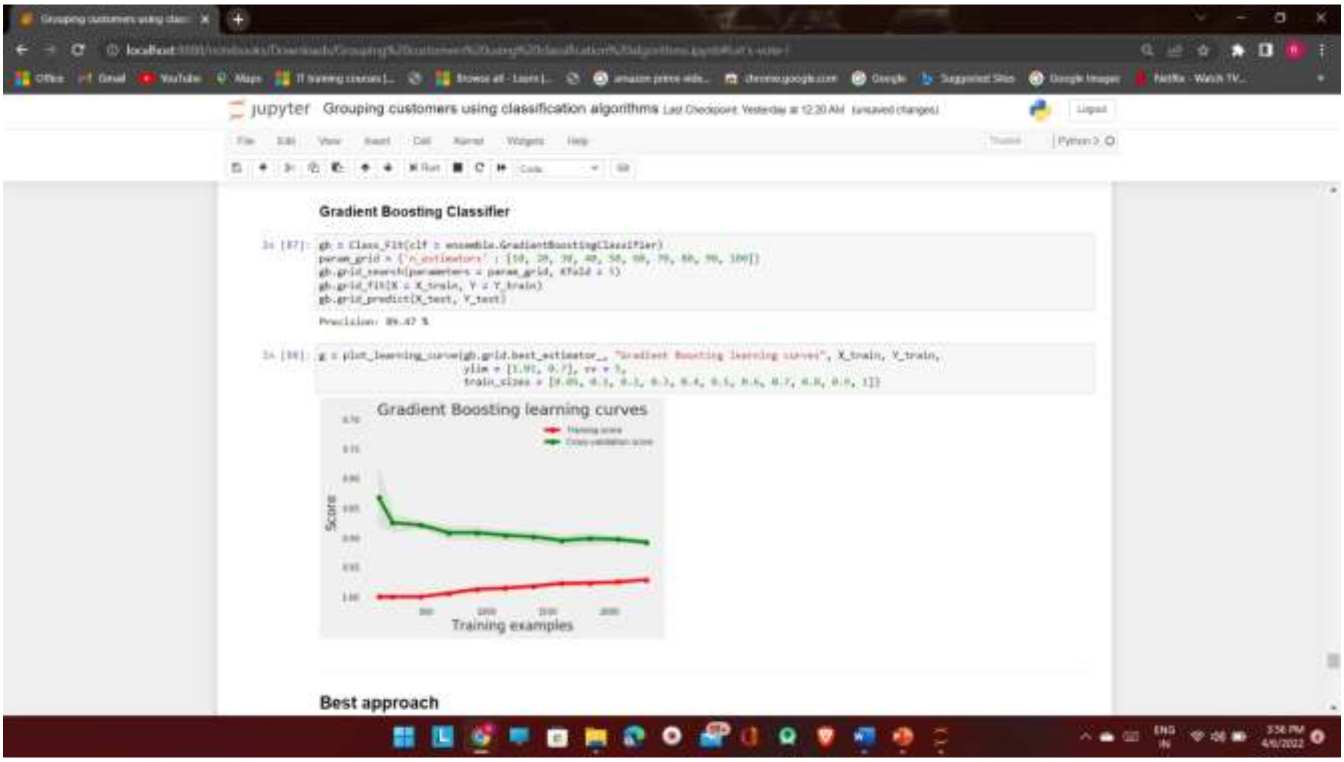
AdaBoost Classifier

```
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_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

```
In [87]: gb = Class_Fit(clf = ensemble.GradientBoostingClassifier)
```

Figure 8.20 : Accuracy of the Ada boost


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Logout

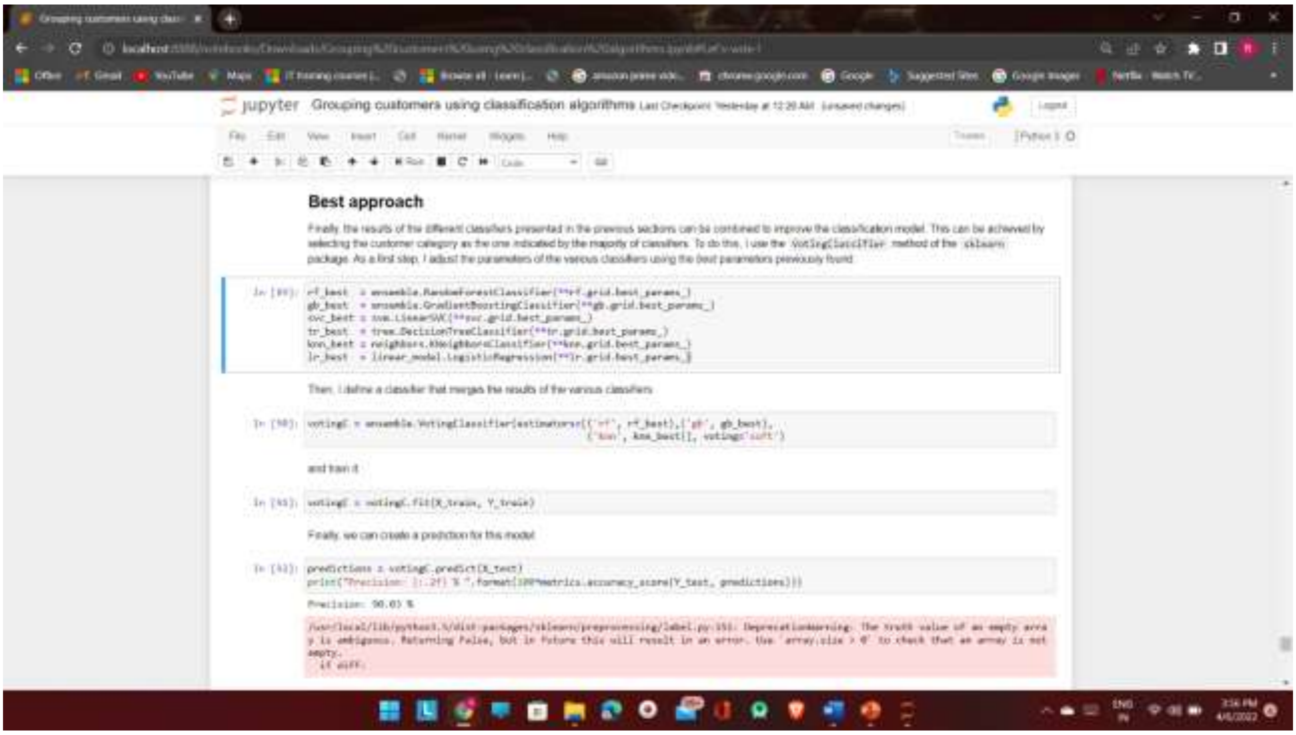
Gradient Boosting Classifier

```
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_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])
```

Best approach

Figure 8.21 : Accuracy of the XGBOOST


jupyter
Grouping customers using classification algorithms Last Checkpoint: Yesterday at 12:20 AM (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
Logout

Best approach

Finally, the results of the different classifiers presented in the previous sections can be combined to improve the classification model. This can be achieved by selecting the customer category as the one indicated by the majority of classifiers. To do this, I use the `VotingClassifier` method of the `sklearn` package. As a first step, I adjust the parameters of the various classifiers using the best parameters previously found:

```
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_)
```

Then, I define a classifier that merges the results of the various classifiers:

```
In [90]: votingC = ensemble.VotingClassifier(estimators=[('rf', rf_best),('gb', gb_best),
                                                         ('knn', knn_best)], voting='soft')
```

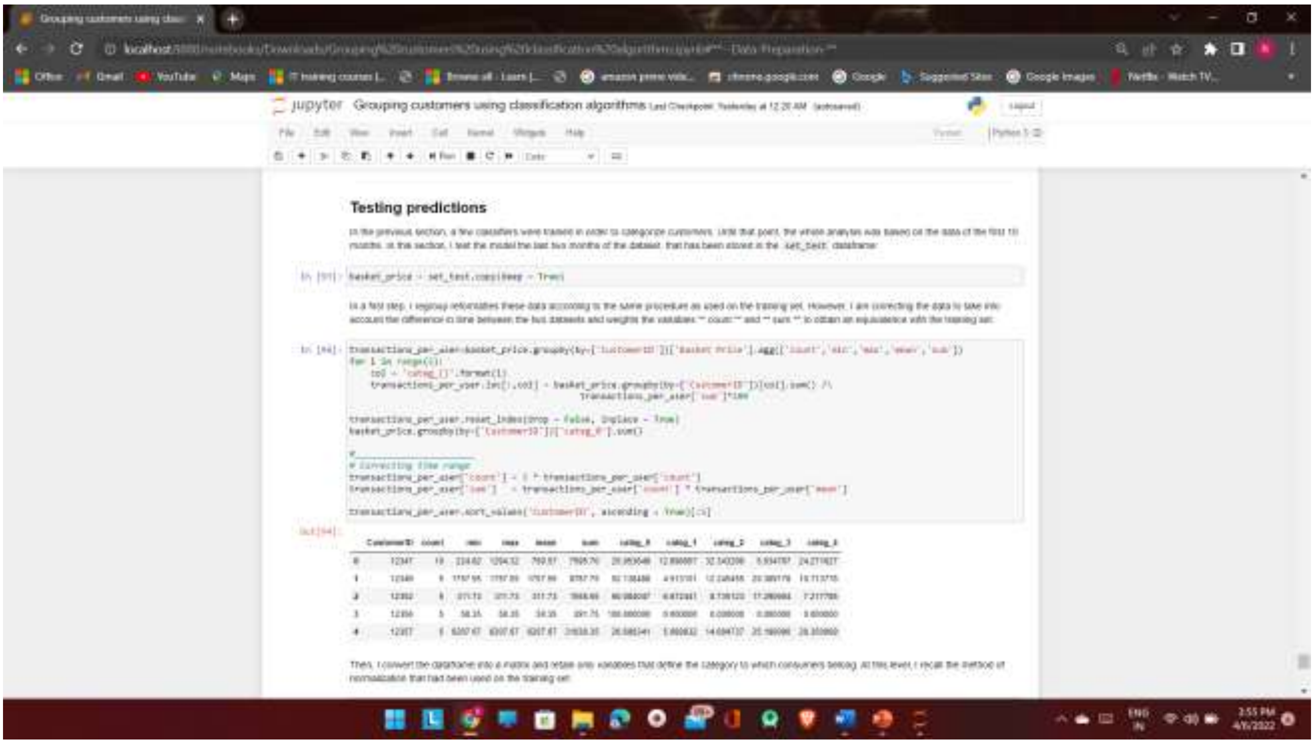
and train it:

```
In [91]: votingC = votingC.fit(X_train, Y_train)
```

Finally, we can create a prediction for this model:

```
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: DeprecationWarning: The truth value of an empty arra
         y is ambiguous. Returning False, but in future this will result in an error. Use `array.size > 0` to check that an array is not
         empty.
           if diff:
```

Figure 8.22 : Accuracy of the Ensemble model in the training phase



Testing predictions

In the previous section, a few classifiers were trained in order to categorize customers. Until that point, the whole analysis was based on the data of the first 10 months. In this section, I test the model the last two months of the dataset, that has been stored in the `set_test` dataframe:

```
In [91]: basket_price = set_test.copy(deep = True)
```

In a first step, I regroup reformattes these data according to the same procedure as used on the training set. However, I am correcting the data to take into account the difference in time between the two datasets and weights the variables ** count ** and ** sum ** to obtain an equivalence with the training set:

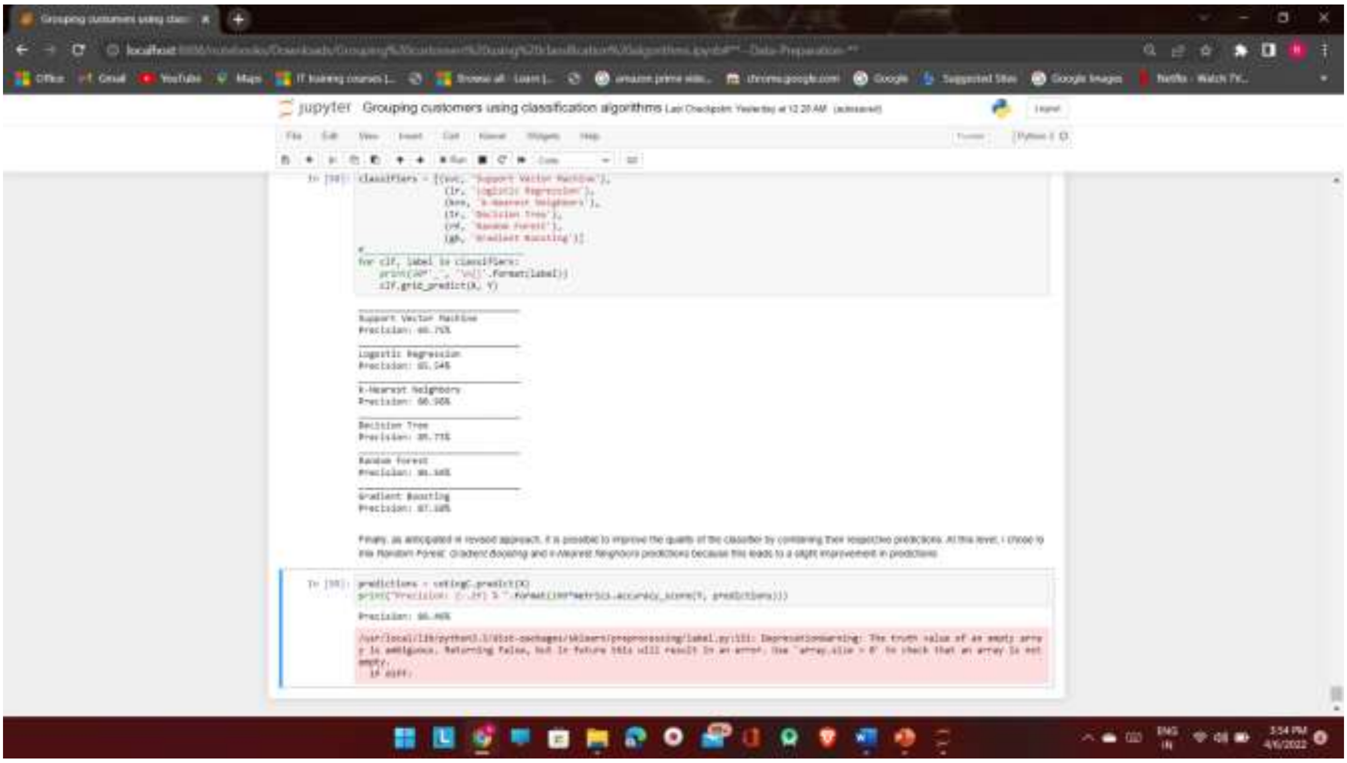
```
In [94]: transactions_per_user=basket_price.groupby(by=['CustomerID'])['Basket Price'].agg(['count','min','max','mean','sum'])
         for i in range(5):
             col = 'categ_{}'.format(i)
             transactions_per_user.loc[:,col] = basket_price.groupby(by=['CustomerID'])[col].sum() /\
                                                 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]
```

Out[94]:

	CustomerID	count	min	max	mean	sum	categ_0	categ_1	categ_2	categ_3	categ_4
0	12347	10	224.82	1294.32	759.57	7595.70	29.953648	12.890087	32.343208	5.934787	24.271827
1	12349	5	1757.55	1757.55	1757.55	8787.75	52.138488	4.913101	12.245455	20.385178	10.713715
2	12352	5	311.73	311.73	311.73	1558.65	60.084087	6.872441	8.738123	17.280684	7.217785
3	12356	5	58.35	58.35	58.35	291.75	100.000000	0.000000	0.000000	0.000000	0.000000
4	12357	5	6207.67	6207.67	6207.67	31038.35	26.585341	5.860832	14.684737	25.158086	28.355868

Then, I convert the dataframe into a matrix and retain only variables that define the category to which consumers belong. At this level, I recall the method of normalization that had been used on the training set:

Figure 8.23 : Testing phase where the data is took from the last 2 months transactions of dataset



```
In [98]: classifiers = [(svc, 'Support Vector Machine'),
                        (lr, 'Logistic 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: 86.75%
______________________________
Logistic Regression
Precision: 85.54%
______________________________
k-Nearest Neighbors
Precision: 80.90%
______________________________
Decision Tree
Precision: 85.71%
______________________________
Random Forest
Precision: 86.56%
______________________________
Gradient Boosting
Precision: 87.10%
```

Finally, as anticipated in revised approach, it is possible to improve the quality of the classifier by combining their respective predictions. At this level, I chose to mix Random Forest, Gradient Boosting and k-Nearest Neighbors predictions because this leads to a slight improvement in predictions:

```
In [99]: predictions = votingC.predict(X)
         print("Precision: {:.2f} % ".format(100*metrics.accuracy_score(Y, predictions)))

Precision: 86.46%

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

Figure 8.24 : Accuracy of the all the algorithms and the ensemble model in the testing phase.

X. CONCLUSION

Since our database was inconsistent as it is a real time data , we need to have a good computing power to run the billions of data. At last we provided the testing phase results where the ML model achieved about 90% accuracy. This is achieved due to the ensemble model learning where it chose the best parameters from each classification algorithm and used as the parameter. The visualization of the clusters is done by the word clouds in the boxes such all the customers who belong to one category has similar interests for the products. In the training phase we can observe that XG boost gave the more accuracy comparatively to other algorithms.

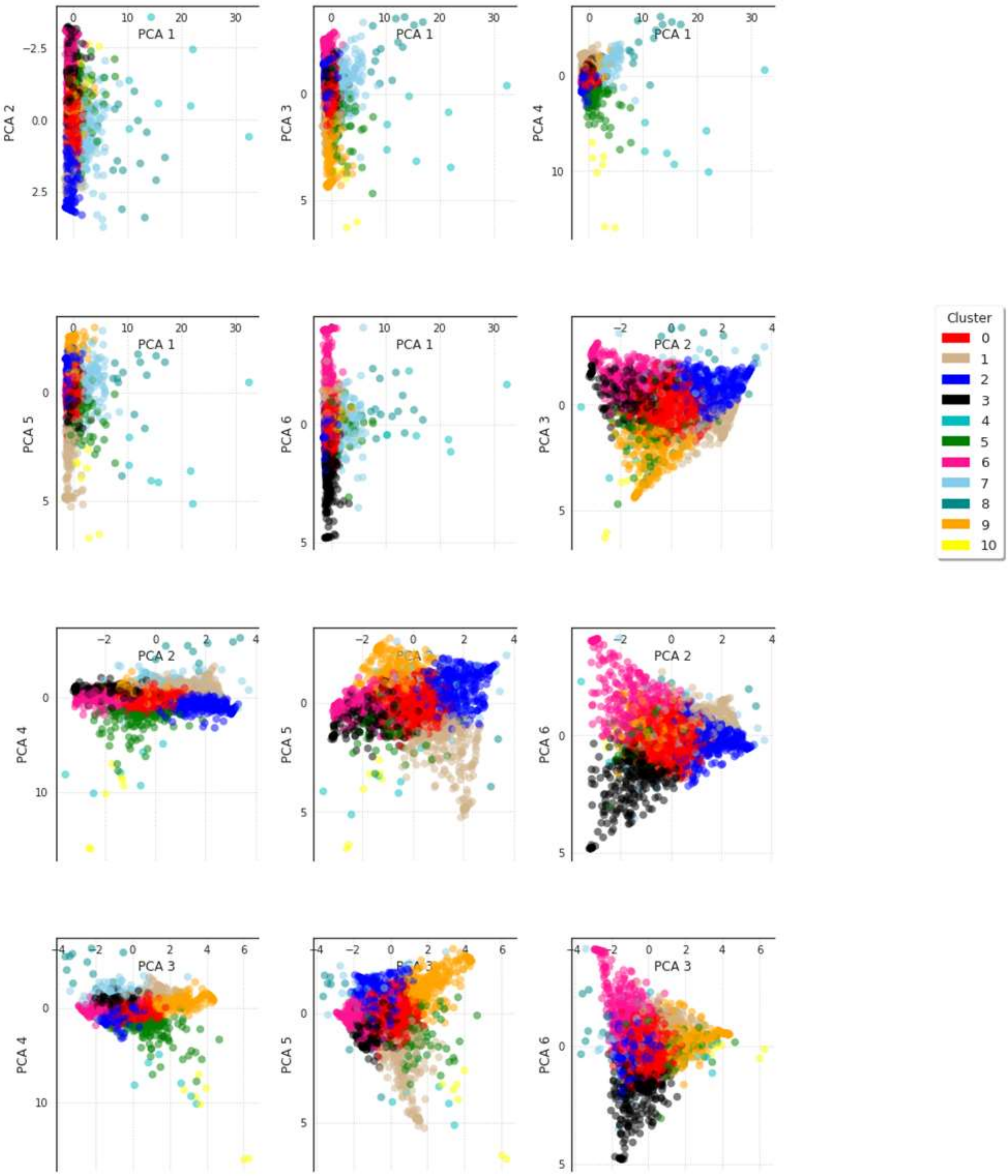


Figure: Displaying all the categories customers.

XI. REFERENCES

[1] Blanchard, Tommy. Bhatnagar, Pranshu. Behera, Trash. (2019). Marketing Analytics Scientific Data: Achieve your marketing objectives with Python's data analytics capabilities. S.l: Packt printing is limited [2] Griva, A., Bardaki, C., Pramatari, K., Papakiriakopoulos, D. (2018). Sales business analysis: Customer categories use market basket data. Systems Expert Systems, 100, 1-16.

[3] Hong, T., Kim, E. (2011). It separates consumers from online stores based on factors that affect the customer's intention to purchase. Expert System Applications, 39 (2), 2127-2131.

[4] Hwang, Y. H. (2019). Hands-on Advertising Science Data: Develop your machine learning marketing strategies… using python and r. S.l: Packt printing is limited

[5] Puwanenthiren Premkanth, - Market Classification and Its Impact on Customer Satisfaction and Special Reference to the Commercial Bank of Ceylon PLC.‖ Global Journal of Management and Business Publisher Research: Global Magazenals Inc. (USA). 2012. Print ISSN: 0975-5853. Volume 12 Issue 1.

[6] Puwanenthiren Premkanth, - Market Classification and Its Impact on Customer Satisfaction and Special Reference to the Commercial Bank of Ceylon PLC.‖ Global Journal of Management and Business Publisher Research: Global Magazenals Inc. (USA). 2012. Print ISSN: 0975-5853. Volume 12 Issue 1.

[7] Sulekha Goyat. "The basis of market segmentation: a critical review of the literature. European Journal of Business and Management www.iiste.org. 2011. ISSN 2222-1905 (Paper) ISSN 2222-2839 (Online). Vol 3, No.9, 2011

[8] By Jerry W Thomas. 2007. Accessed at: www.decisionanalyst.com on July 12, 2015.

[9] T.Nelson Gnanaraj, Dr.K.Ramesh Kumar N.Monica. AnuManufactured cluster analysis using a new algorithm from structured and unstructured data. International Journal of Advances in Computer Science and Technology. 2007. Volume 3, No.2.

[10] McKinsey Global Institute. Big data. The next frontier is creativity, competition and productivity. 2011. Accessed at: www.mckinsey.com/mgi on July 14, 2015.

[11] Jean Yan. - Big Data, Big Opportunities- Domains of Data.gov: Promote, lead, contribute, and collaborate in the big data era. 2013. Retrieved from http://www.meritalk.com/pdfs/bdx/bdxwhitepaper-090413.pdf July 14, 2015.

[12] A.K. Jain, M.N. Murty and P.J. Flynn.‖ Data Integration: A Review‖. ACM Computer Research. 1999. Vol. 31, No. 3.

[13] Vishish R. Patel1 and Rupa G. Mehta. MpImpact for External Removal and Standard Procedures for JCSI International International Science Issues Issues, Vol. 8, Appeals 5, No 2, September 2011 ISSN (Online): 1694-0814 7

[14] Jayant Tikmani, Sudhanshu Tiwari, Sujata Khedkar "Telecom Customer Classification Based on Group Analysis of K-methods", JIRCCE, Year: 2015.

[15] Vaishali R. Patel and Rupa G. Mehta “Impact of Outlier Removal and Normalization Approach in Modified k-Mean