UNIVERSITÀ DI PISA

DATA MINING PROJECT

A.A. 2019-2020

I Contachilometri team

Marco Mazzei (546816)

Andrea Cardia (540370)

Carlo Paladino (537650)

Giulia Ferro (597681)

SUMMARY

1. DATA UNDERSTANDING

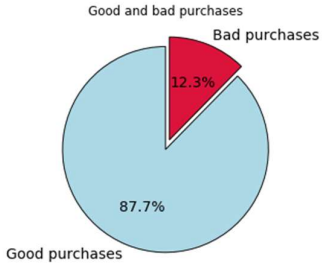
a. Data semantic and distribution of variables

Carvana is an American online start-up that has as core business the retail of used cars. In this business, the ability to predict if a car will be a good or bad purchase brings an essential competitive advantage. This prediction can prevent the risk of unsold cars which causes damages to the prosperity of the company. The aim of this project is to prevent those possible damages analysing the data given by the start-up.
Carvana published a dataset containing 58386 instances, with 32 independent variables and a dependent variable (IsBadBuy), plus another dataset of 14597 instances for testing with the same variables.

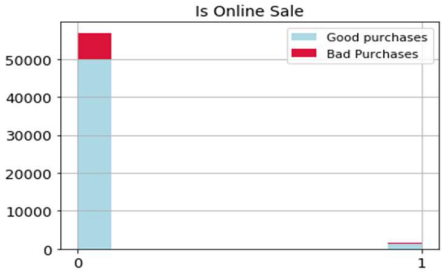
```
RangeIndex: 58386 entries, 0 to 58385
Data columns (total 34 columns):
RefId                             58386 non-null int64
IsBadBuy                          58386 non-null int64
PurchDate                         58386 non-null object
Auction                           58386 non-null object
VehYear                           58386 non-null int64
VehicleAge                        58386 non-null int64
Make                              58386 non-null object
Model                             58386 non-null object
Trim                              56475 non-null object
SubModel                          58379 non-null object
Color                             58379 non-null object
Transmission                      58378 non-null object
WheelTypeID                       55813 non-null float64
WheelType                         55809 non-null object
VehOdo                            58386 non-null int64
Nationality                       58382 non-null object
Size                              58382 non-null object
TopThreeAmericanName              58382 non-null object
MMRAcquisitionAuctionAveragePrice 58373 non-null float64
MMRAcquisitionAuctionCleanPrice   58373 non-null float64
MMRAcquisitionRetailAveragePrice  58373 non-null float64
MMRAcquisitonRetailCleanPrice     58373 non-null float64
MMRCurrentAuctionAveragePrice     58141 non-null float64
MMRCurrentAuctionCleanPrice       58141 non-null float64
MMRCurrentRetailAveragePrice      58141 non-null float64
MMRCurrentRetailCleanPrice        58141 non-null float64
PRIMEUNIT                         2683 non-null object
AUCGUART                          2683 non-null object
BYRNO                             58386 non-null int64
VNZIP1                            58386 non-null int64
VNST                              58386 non-null object
VehBCost                          58386 non-null float64
IsOnlineSale                      58386 non-null int64
WarrantyCost                      58386 non-null int64
dtypes: float64(10), int64(9), object(15)
```

In addition to these two datasets was provided a file with the description of each variable. There are fifteen variables that are represented by categorical attributes and nineteen numerical variables among which, nine are integers and the remaining ten are float values, like can be seen in the previous list.
To better understand the meaning of the variables, we decide to divide them by their informational purpose and to show their distribution. In the dataset there are two unique identifiers: “RefID” that is assigned to each record and “BYRNO” that is assigned to every buyer. There are two binary variables: “IsBadBuy” and “IsOnlineSale”. The first one is the only dependent variable. If it assumes the value 1 it means that the purchase

was a bad one, otherwise 0 represents a good deal. As we can see from the following pie chart, the two classes are very unbalanced because the majority of the purchases were good ones. More precisely, there are 51178 good purchases and 7208 bad ones.

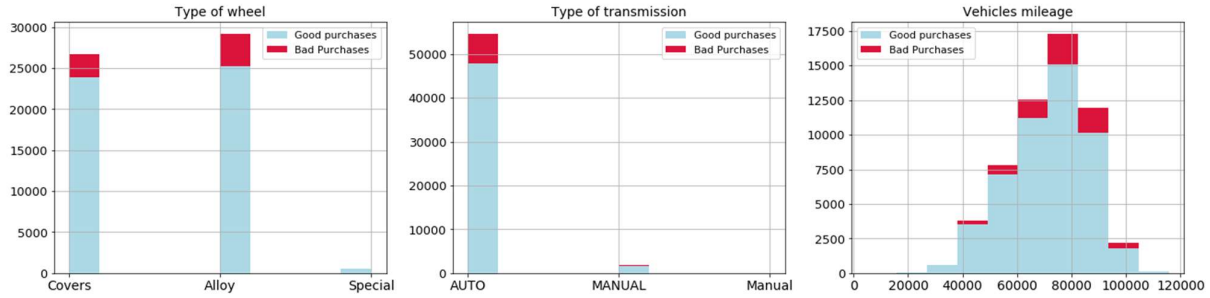


The second binary variable, "IsOnlineSale", identifies with the number 1 the online purchases and 0 the other type of purchases. Even if the majority aren't online sale (97%), the proportion of bad purchases is almost equally present in both kind of purchase method.

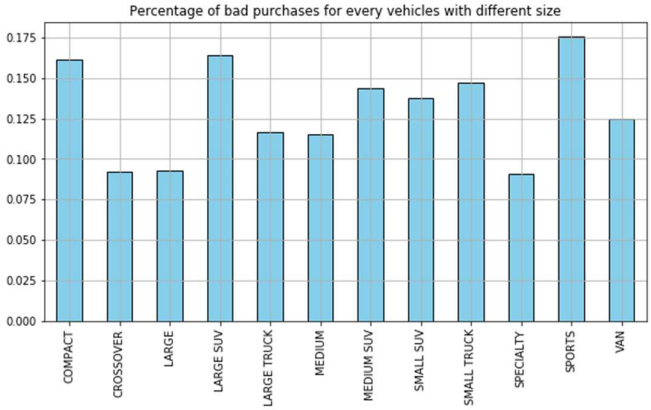


IsBadBuy	0	1
IsOnlineSale		
0	49852	7035
1	1326	173

The information about the technical features of the vehicles are given by five different variables: "Trim", "Color", "WheelType", "Transmission" and "VehOdo". They indicate respectively level of optional

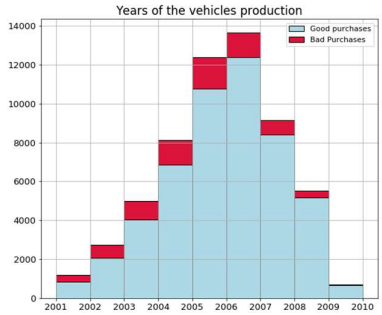


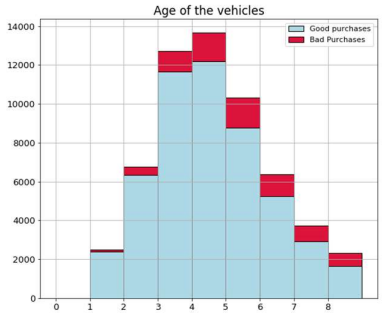
equipment (131 different values), colour, type of wheel, kind of transmission (automatic or manual) and odometer reading. "WheelType" is linked with "WheelTypeID" which is the ID of the type of wheels.
There are other informations to identify the kind of car, namely "Model", "SubModel" and "Size" which respectively specify the kind of model, submodel and the category of the vehicle (e.i. Compact, SUV…).

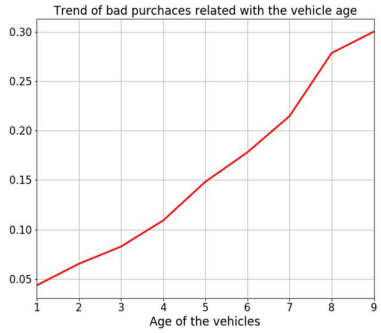


We also have three temporal information. The feature called “purchDate” describes the date, month and year when the vehicle was purchased and “VehYear” shows the production year. From these two features, we get “VehicleAge” which indicates the age of the vehicle. The following histograms show the distribution of these two last variables. We can see that the majority of the vehicles are between three and five years old and were made between 2005 and 2007.

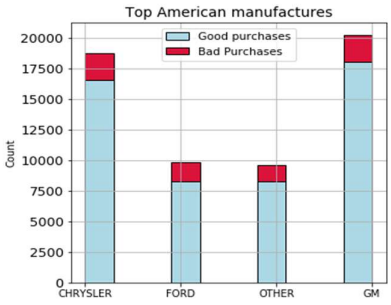
As expected, in the last of the following graphs, we can see that the percentage of the bad purchases grows when the years of the vehicle increase, so it is very likely to select a bad purchase with older cars.

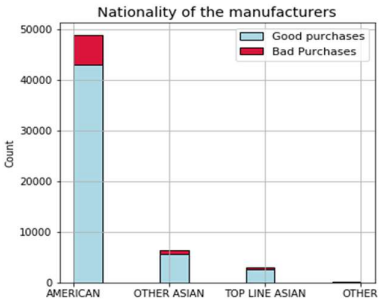




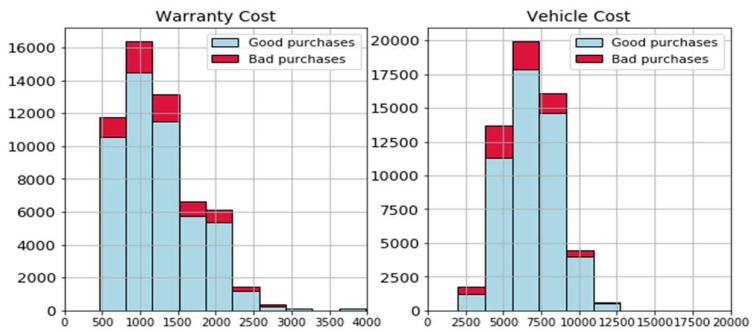


Regarding the manufacturer there are three features: “Make” which represent the 32 car manufacturers, “TopThreeAmericanName” which shows if a car is made by one of the main American manufacturers (GM, Ford, Chrysler) or not (Others) and “Nationality” that is about the country of the manufacturers. We also have information about the auction provider at which the vehicle was purchased with “Auction” and the place where that happened with the indication about the state and the zip code given by “VNST” and “VNZIP”.

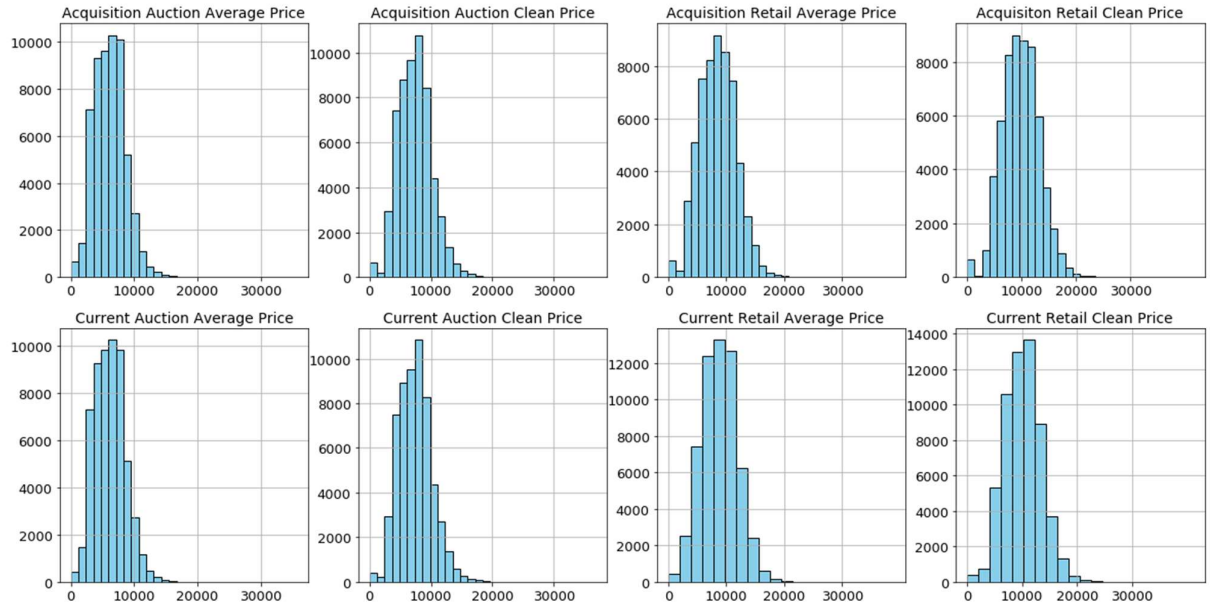




Regarding the prices, we have ten different variables. “VehBCost” indicates the cost of the car and the “WarrantyCost” is the price of the warranty. They have the following distributions.



The other prices are expressed by the following variables: “MMRAcquisitionAuctionAveragePrice”, “MMRAcquisitionAuctionCleanPrice”,“MMRAcquisitionRetailAveragePrice”, “MMRAcquisitonRetailCleanPrice”,“MMRCurrentAuctionAveragePrice”, “MMRCurrentAuctionCleanPrice”, “MMRCurrentRetailAveragePrice”, “MMRCurrentRetailCleanPrice”.



All the variables with “Acquisition” have information about the price at the moment of the purchase (first row of graphs) and all the variables whit “Current” show the value of the vehicle nowadays (second row of graphs). The prices are also divided if the car was purchased in an auction or retail and if the vehicle was on average conditions (average price) or if it was in very good conditions (clean price). We can see that the clean price is higher of the average from the differences of the graphs because the distributions with clean price are a little more shift on the right where the prices are higher.
The last two variables are “PRIMEUNIT” and “AUCGUART”, the first one defines if a vehicle has a higher demand than the others, and the second one is about the level of guarantee provided by the auction for the vehicle.

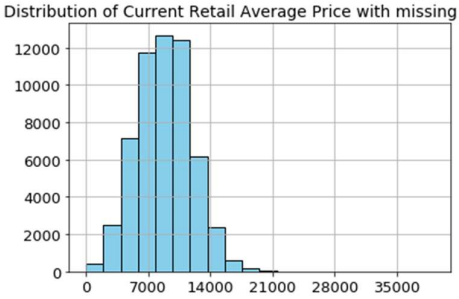
b. Outliers and missing values removal

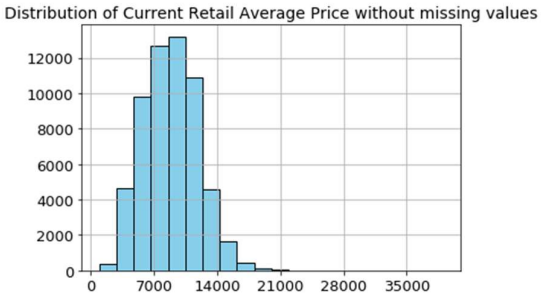
Like the majority of the datasets, even this one, has some missing values. In the following table we can see the total number of the missing values for each variable.

RefId	0	TopThreeAmericanName	4
IsBadBuy	0	MMRAcquisitionAuctionAveragePrice	13
PurchDate	0	MMRAcquisitionAuctionCleanPrice	13
Auction	0	MMRAcquisitionRetailAveragePrice	13
VehYear	0	MMRAcquisitonRetailCleanPrice	13
VehicleAge	0	MMRCurrentAuctionAveragePrice	245
Make	0	MMRCurrentAuctionCleanPrice	245
Model	0	MMRCurrentRetailAveragePrice	245
Trim	1911	MMRCurrentRetailCleanPrice	245
SubModel	7	PRIMEUNIT	55703
Color	7	AUCGUART	55703
Transmission	8	BYRNO	0
WheelTypeID	2573	VNZIP1	0
WheelType	2577	VNST	0
VehOdo	0	VehBCost	0
Nationality	4	IsOnlineSale	0
Size	4	WarrantyCost	0

In thes steps we substitute or delate the missing values and some outliers.

- The missing values of “TopThreeAmericanName” and “Nationality” were substituted manually using “Make” as a reference.
- The two variables "PRIMEUNIT" and "AUCGUART" have the 95.4% of the attributes null, so we decided to delete this features from the dataset because they are not useful for the study.
- For the prices we decided to substitute missing value and some outliers which had 1 and 0 as values. We started by replacing these 0 and 1 values with ‘np.nan’ to define them as missing value and later we sobtitute them. To replace all these missing values we used the mean done by grouping between ‘Make’, ‘Model’ and ‘SubModel’ to have more precise results. Doing so we subtituted the most of the values and the remaining ones were substituted using the mean done by grouping just between ‘Make’ and ‘Model’. We choose to use the mean instead of the median because there wasn’t significant alteration in the distribution before and after the changes, like can be seen in the following plots.

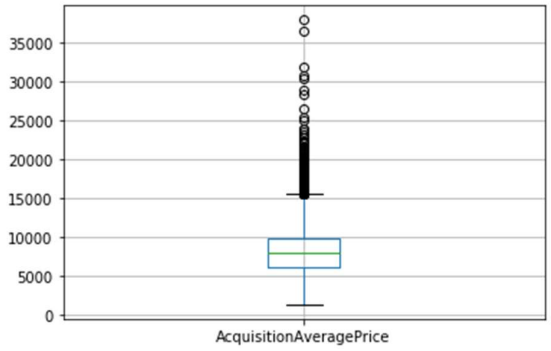


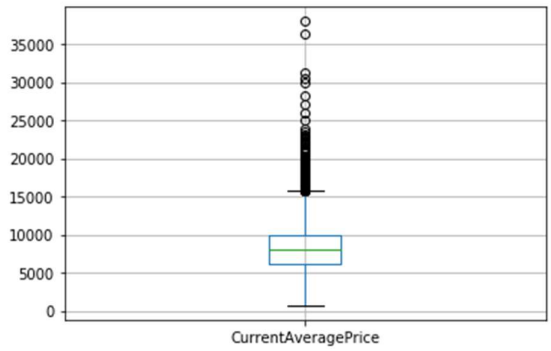


- To replace the missing values that are in ‘Trim’, we transform the attributes of the variable in a string format, because otherwise there were problems with some integer values. Then we use the mode made between all the vehicles with the same manufacturer, model and sub model to have more accurate results. This last step was used also to replace the missing values of the variable ‘WheelType’. This last variable was also used with its missing value that were replaced with a special character because it gives some unexpected information.
- We decide to delete the rows that contains missing values in the variables ‘Color’, ‘Submodel’ and ‘Size’ because there were only few rows with multiple missing values.
- In the variable ‘Transmission’ we use the mode between vehicle with the same model and the same manufacturers to substitute the missing values.

c. Variable transformation

The last step to improve the organization of the dataset is about variable transformations. First of all, we decide to delete the variable that are unique identifiers of a specific attribute, so they aren’t useful for the study. That’s why we delete 'BYRNO', 'RefId' and 'WheelTypeID'. To improve the data for the analysis we create some new variables. ‘MilesYear’, that represents the amount of miles made by a vehicle in a year. To create ‘MilesYear’ we divide the total amount of mile (given by ‘VehOdo’) with the age of the vehicle (‘VehicleAge’). From ‘PurchDate’ we create two different variables: one with the month of the purchase (‘Month’) and the other with the day (‘Day’). Another variable called ‘VehBCostVehicleAge’ is given by the ratio between ‘VehBCost’ and ‘VehicleAge’. From the prices we define two new features: one with the average between all the acquisition prices ('AcquisitionAveragePrice'), and the other with all the current prices (‘CurrentAveragePrice’). We check their boxplot and as a result we have that the 50% of the observation are between 600 and 1000 for both graphs. All the values higher than 1500 are considered outliers.





To create relations between the prices, we made 4 new variables with the differences between every current and acquisition prices, other 8 variables with the differences between 'VehBCost' and the current and acquisition prices. All those new features will be very useful later on during the classification, that's why we created them also in the test set.
In the end we solved another problem. The variable 'Transmission' presented some value as 'manual' instead of 'MANUAL', so we change these two different characters to have all the features divided in the right two classes 'MANUAL' and 'AUTOMATIC'.

d. Correlation

To better understand the relations between the variables we analyse the correlation matrix. Even if there aren't any specific unexpected correlations, some results confirmed our expectations. For example, there is a negative correlation between all the prices variables and 'VehAge' because if the vehicles are older, they will probably cost less (the same but opposite principle between prices variables and 'VehYear'). The other interesting correlation can be found between 'Warranty' and 'VehOdo' where a positive correlation, even if not very strong (0.4), implies that if the mileage of the vehicle increases then also the warranty cost will increase.

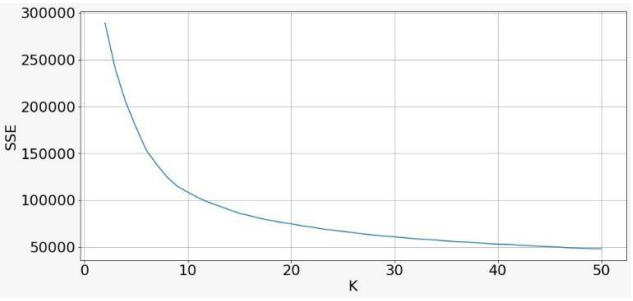
2. CLUSTERING

a. Variable choice

This kind of dataset has a particular distribution of data that doesn't allow to find define clusters with a low SSE value and low silhouette. That's why we decide to use this algorithm with fewer variables that are very useful later in the classification task, so it can be useful to have some more information about them. These variables are: 'VehOdo', 'VehBCostVehicleAge', "DiffAcquisition_CurrentRetailAveragePrice" obtained by "MMRAcquisitionRetailAveragePrice"–"MMRCurrentRetailAveragePrice", "DiffAcquisition_CurrentAuctionAveragePrice" given by the difference between "MMRAcquisitionAuctionCleanPrice" and "MMRCurrentAuctionCleanPrice". The only categorical attribute used is 'WheelType' that is transformed in numerical one by assigning a number to every feature (Alloy = 1, Cover = 2, Special = 3 and the missing values are equal to -5).
The first algorithm used is k-means and to choose the right number of clusters (k) we check the knee of the SSE.

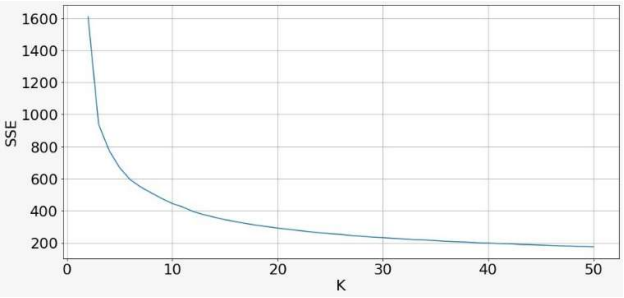
Normalization	K	Silhouette Score	SSE
Z-SCORE	6	0.249	153342
Z-SCORE	7	0.270	138137
Z-SCORE	8	0.271	124782
Z-SCORE	9	0.268	114817
Z-SCORE	10	0.213	108604
Z-SCORE	11	0.227	103188
Z-SCORE	12	0.210	98599
Z-SCORE	13	0.222	93775
Z-SCORE	14	0.217	89832

SSE curve with Zscore normalization



MinMax	6	0.310	593
MinMax	7	0.304	547
MinMax	8	0.310	511
MinMax	9	0.312	476
MinMax	10	0.300	444
MinMax	11	0.288	413
MinMax	12	0.290	394
MinMax	13	0.278	375
MinMax	14	0.264	359

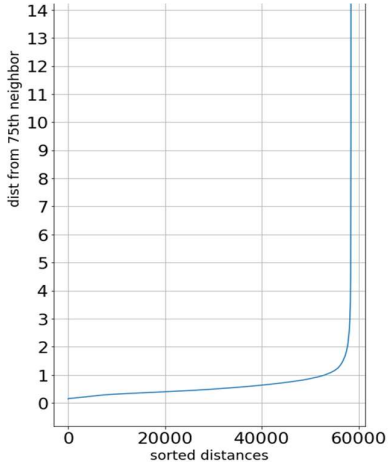
SSE curve with MinMax normalization



The two previous graphs show the SSE curve for both the normalization MinMax and Zscore. Even if the MinMax normalization present lower SSE values, we decide to use the Zscore normalization because it is more precise and as metric the Euclidean distance. In the table there are instead all the different values of SSE and silhouette regarding various number of cluster k.

b. Outlier detection and removal with Dbscan

These results can be improved because in the dataset there still are the outliers and to remove them is used Dbscan algorithm. To find the best parameters of min sample and eps (radius), we tried different values changing the number of min sample from 5 to 95 and choosing each time the value on the knee of the curve given by the following graph.



We save all the values in the next table and we choose the one with the higher silhouette which is equal to 0,491.

Normalization	MinSamples	EPS	Silhouette	K	Noise Points
Z-SCORE	5	0.9	0.392	13	634
Z-SCORE	10	0.95	0.469	6	777
Z-SCORE	15	1	0.473	4	841
Z-SCORE	20	1.05	0.489	3	827
Z-SCORE	25	1.1	0.491	2	819
Z-SCORE	30	1.1	0.490	2	909
Z-SCORE	**35**	**1.2**	**0.491**	**2**	**744**
Z-SCORE	40	1.2	0.478	2	797
Z-SCORE	45	1.2	0.489	2	896
Z-SCORE	50	1.3	0.489	2	739
Z-SCORE	55	1.4	0.489	2	625
Z-SCORE	60	1.45	0.489	2	591
Z-SCORE	65	1.5	0.500	2	549
Z-SCORE	70	1.55	0.490	2	529

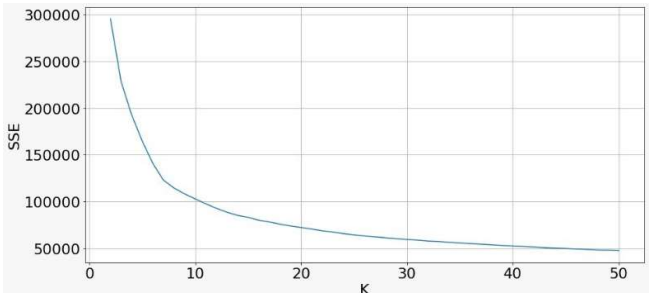
Z-SCORE	75	1.6	0.490	2	485
Z-SCORE	80	1.65	0.490	2	462
Z-SCORE	85	1.7	0.490	2	446
Z-SCORE	90	1.7	0.491	2	457
Z-SCORE	95	1.75	0.491	2	442
MinMax	5	0.04	0.005	33	1189
MinMax	10	0.04	0.064	13	1891
MinMax	15	0.05	0.169	6	1261
MinMax	20	0.05	0.085	10	1472
MinMax	25	0.06	0.234	4	965
MinMax	30	0.06	0.175	6	1090
MinMax	35	0.065	0.209	5	951
MinMax	40	0.065	0.225	4	1078
MinMax	45	0.07	0.228	4	906
MinMax	50	0.07	0.227	4	961
MinMax	55	0.075	0.229	4	818
MinMax	60	0.075	0.228	4	862
MinMax	65	0.08	0.230	4	731
MinMax	70	0.08	0.227	4	761
MinMax	75	0.085	0.232	4	622
MinMax	80	0.09	0.235	4	537
MinMax	85	0.09	0.234	4	558
MinMax	90	0.095	0.236	4	465
MinMax	95	0.095	0.233	4	486

c. Choice of best parameters for Kmeans and Dbscan

After that the removal of the outliers we recalculate the values of the previous tables and also the SSE curve.

K	Silhouette Score	SSE
6	0.251	140684
7	**0.274**	**122817**
8	0.250	114379
9	0.236	108120
10	0.231	97482
11	0.221	92649
12	0.225	88259
13	0.227	85059
14	0.224	82044

SSE curve with Zscore normalization and without ouliers



From the table below we can see that there is a little improvement in Kmeans algorithm because the silhouette is higher and the error is lower. Also the SSE curve is changed and it is lower. Now we can choose the best k by looking at the values on the table and the knee of the SSE curve. We decide to keep seven 7 k because it has the highest silhouette score. For Dbscan, the new parameters can be seen in the table below. The higher value of silhouette score is now 0.6, so we choose min sample equal to 75 and eps equal to 1.6 as new parameters.

MinSamples	EPS	Silhouette	K	Noise Points
5	0.9	0.341	6	394
10	0.95	0.492	3	472
15	1	0.402	3	458
20	1.05	0.515	2	297
25	1.1	0.542	2	347
30	1.1	0.546	2	507
35	1.2	0.561	2	260
40	1.2	0.566	3	364

45	1.2	0.578	2	460
50	1.3	0.597	2	250
55	1.4	0.608	2	111
60	1.45	0.609	2	97
65	1.5	0.608	2	66
70	1.55	0.608	2	48
75	**1.6**	**0.610**	**2**	**45**
80	1.65	0.609	2	35
85	1.7	0.608	2	29
90	1.7	0.608	2	34
95	1.75	0.607	2	26

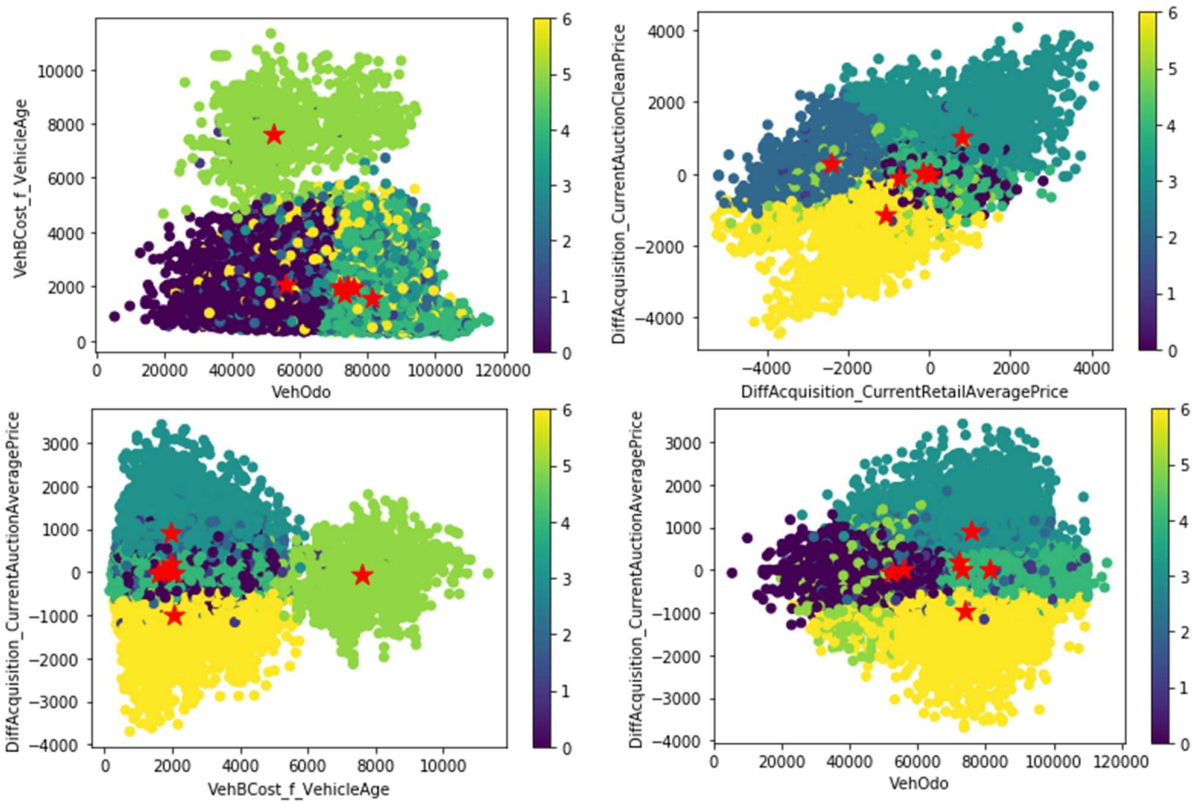
d. Cluster characterization

Now that we have the parameters for Kmeans and Dbscan, we proceed with the analysis of graphs with two variable each (total of 15 graphs) at first with Kmeans then with Dbscan.

With Kmeans the number of elements in each cluster is: cluster 0 = 4399 elements, cluster 1 = 13167, cluster 2 = 2445, cluster 3 = 2355, cluster 4 = 8169, cluster 5 = 20361, cluster 6 = 6733. The centroids that the algorithm found are represented with a red star in the graphs and are:

Centroid	WheelType	VehOdo	DiffAcquisition_CurrentRetailAverage	VehBCost_f_VehicleAge	DiffAcquisition_CurrentAuctionAverage	DiffAcquisition_CurrentAuctionClean
0	1,44	71885,86	-2429,45	1976,61	200,05	306,98
1	1,64	55791,71	10,05	2056,85	-5,76	-27,71
2	-5,00	73116,89	-143,97	1736,03	11,19	15,60
3	1,81	52082,72	-719,28	7609,88	-58,69	-105,43
4	1,41	75903,87	785,92	1943,17	901,94	1008,70
5	1,44	81066,06	15,35	1569,50	-4,27	-23,11
6	1,41	74000,24	-1091,82	2027,53	-979,59	-1120,54

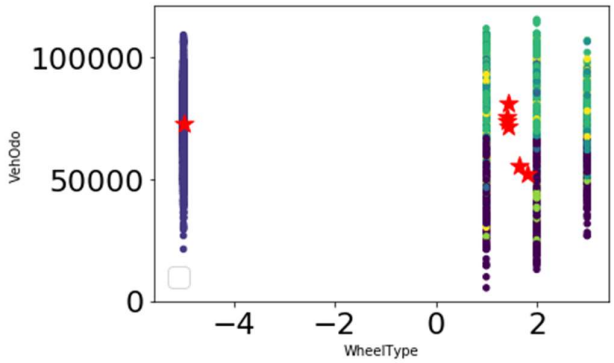
The most representative graphs for Kmeans are the following:



The bar on the side of each graph help us to understand which colour is the cluster. To better understand the composition of this graphs we made the characterization trough the target variable 'IsBadBuy'. The results are the following:

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
Bad purchases	1045	**1792**	321	670	2740	54	506
Good purchases	12123	**653**	4077	7500	17620	2301	6227

The most interesting result is given by the cluster 1 where the number of bad purchases are higher than the number of good ones. From the next graphs we can see that the majority of the missing value of the feature 'WheelType' are grouped here.



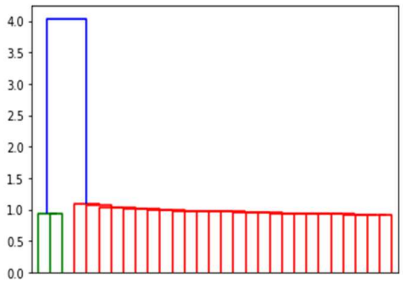
For Dbscan we have only two clusters that are composed one by 55156 elements and the other by 2428 elements. This clusters are very unbalanced and none of the graphs give us new information, even with the characterization with the target variable 'IsBadBuy' doesn't have a lot of insights because the proportion of bad purchases and good one reflects the proportion of the dataset. In fact, in the first cluster there are 49827 good purchases and 5329 bad ones. The second cluster has 1780 good buys and 648 bad ones.

e. Hierarchical algorithm

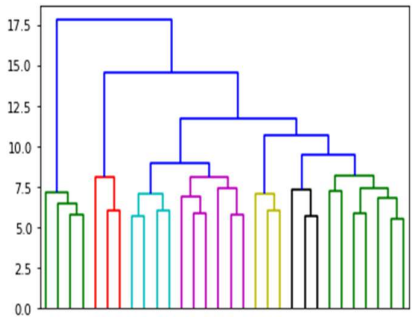
The last clustering algorithm we analyse is the hierarchical. We keep the same attribute that are used for Kmeans and Dbscan, the same normalization Zscore and the Euclidean distance. The outliers were already deleted using Dbscan. To reduce the complexity of the algorithm, we decide to not start from every single point but to make a sample. To do so we used the Kmeans algorithm, with a high number of clusters equal to 11625 (the 20%of the dataset) to get the starting point of the hierarchical.

As clustering techniques, we analyse:

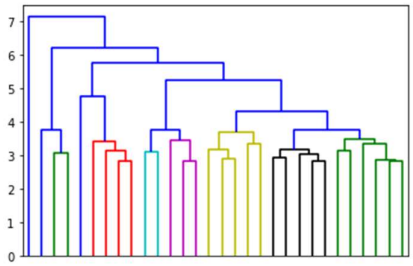
- Single link. Like can be seen in the graph this method isn't appropriate for the structure of our data, that's why we can't obtain a lot of information.



- Complete link. This method seems to be more suitable for the dataset. We decide to have a representation of the same number of clusters found by Kmeans namely seven. To do so we set the value of the threshold equal to 8,3.



- Average link. With this method we applied the same strategy used for Complete link, but this time to get 7 clusters the threshold is set equal to 3.8.



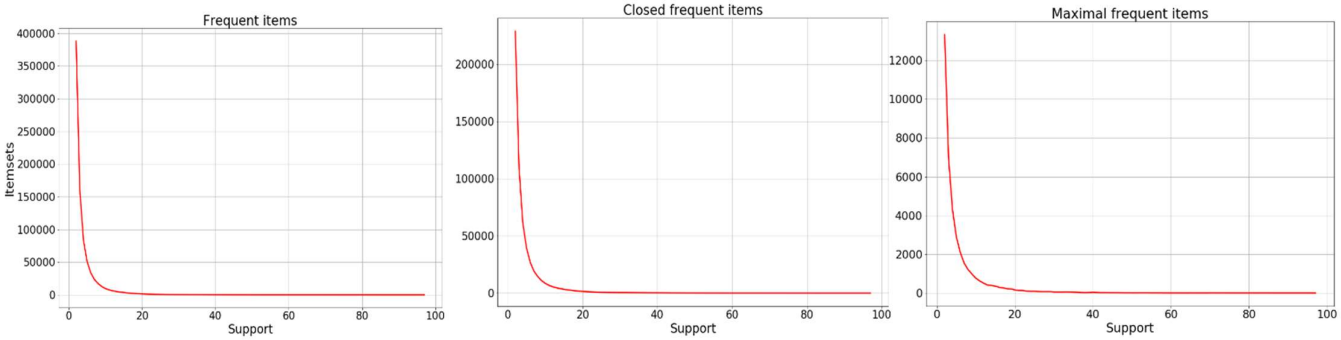
The most useful clustering algorithm for the informational purpose is the Kmeans. The Cluster 1 includes all the null values of the variable 'WheelType' and from the characterization we know that the number of bad purchases is higher in this cluster than the good ones. On the other hand, the best algorithm to eliminate the missing values is Dbscan.

3. ASSOCIATION RULES MINING

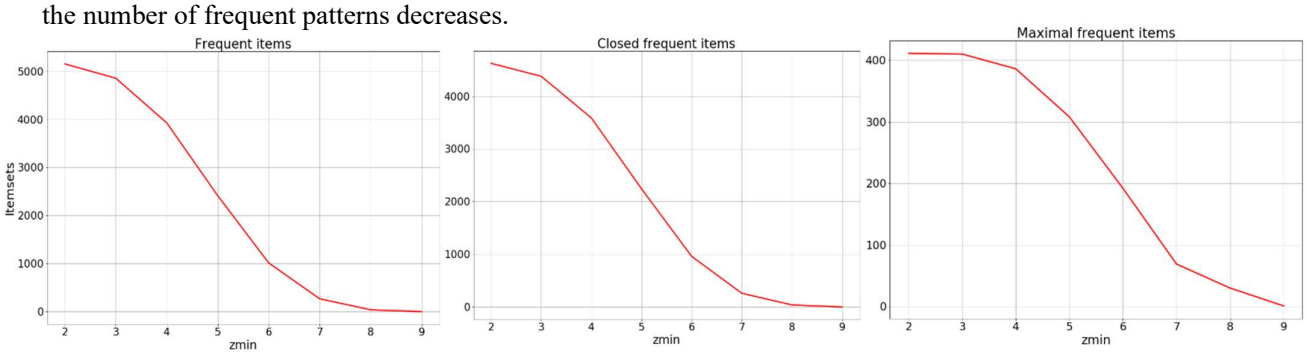
a. Frequent pattern extraction with different parameters

Before starting the association rules the variables have to be prepared for the task. All the continuous features are transformed in discrete ones through the definition of six ranges for: 'WarrantyCost', 'VehBCost', 'AcquisitionAveragePrice', 'CurrentAveragePrice', 'MilesPerYear', 'VehOdo'. The values of the two binary features, 'IsBadBuy' and 'IsOnlineSale', were substituted by their values (if 'IsBadBuy' equal to 0 the value is 'good', otherwise is 'bad').

For the association rules task is used the Apriori algorithm, which is divided in two steps: frequent pattern detection and rules extraction. In the first step are searched the set of items that occurs very frequently in the data set, to do so two parameters are define: the minimum number of items that have to be in a frequent itemset (zmin) and the support that is a measure of how frequently a pattern is present in the dataset. The second step is about rules extraction, to do so is define also the percentage, another parameter that measure the reliability of a rule. Regarding the first step, to find the frequent itemset we make different attempts changing the parameters values. We start by selecting a fixed value of zmin equal to 2 and for the support a range from 2 to 97 to understand how the number of frequent itemset changes with the increasing of the support. In the following first graph we can see how the total number of frequent itemset decreases with the increasing of the support. In the second graphs are displayed the number of closed itemset, namely without superset with the same frequency. This kind of patterns are a subgroup of the total frequent itemset, that's why they follow the same trend. In the last graphs are represented the maximal itemset, the frequent itemset without frequent superset, which in their turn are a subgroup of the closed frequent itemset.



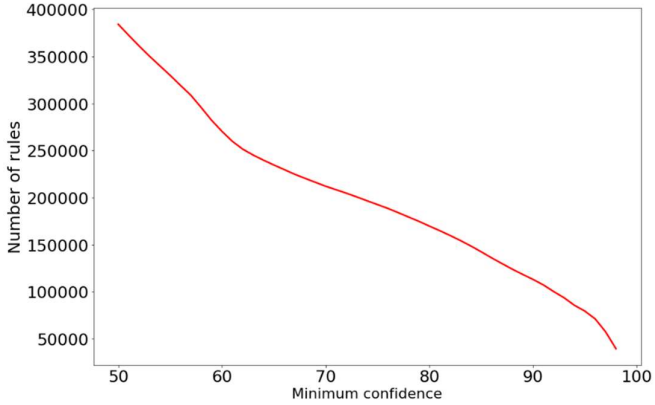
With the given values of zmin, the algorithm can find frequent subset until a support equal to 94. The most frequent itemset, with zmin=2 and support= 94, is composed by 'AUTO' (indicates the kind of transmission of the vehicle) and 'IsOnlineSale' equal to 0 (so it's not an online sale). If the value of the support decreases at 84 the algorithm returns three frequent patterns: the first is the same given before, the second one composed by 'good' (which indicates that it's a good purchase) and 'IsOnlineSale' equal to 0, and the last with 'GOOD'and 'AUTO'. In this first step of the analysis we have frequent itemset with 2 features because zmin is fixed and equal to 2, the number of itemset changes because the value of the support changes. In the following step we switch those two parameters by taking a fixed support equal to 13 (value on the knee of the curve in the previous graphs) and changing the zmin value in range from 2 to 9 because after that there is no frequent itemset. The logic used is the same as before with the identification of the total frequent pattern, the closed and the maximal. To give a more exhaustive view, the following lists show the exact number of frequent itemset starting from a zmin equal to 2 up to 10 (like said before the last is empty because zmin is higher than 9).
Number of frequent itemset [9672, 9266, 7859, 5257, 2470, 741, 125, 10, 0]
Number of closed frequent itemset [8536, 8204, 7041, 4813, 2315, 711, 122, 10, 0]
Number of maximal frequent itemset [768, 760, 720, 634, 419, 186, 45, 10, 0]
The following graphs reflect the values of the lists where we can see that with the increasing of the zmin value the number of frequent patterns decreases.



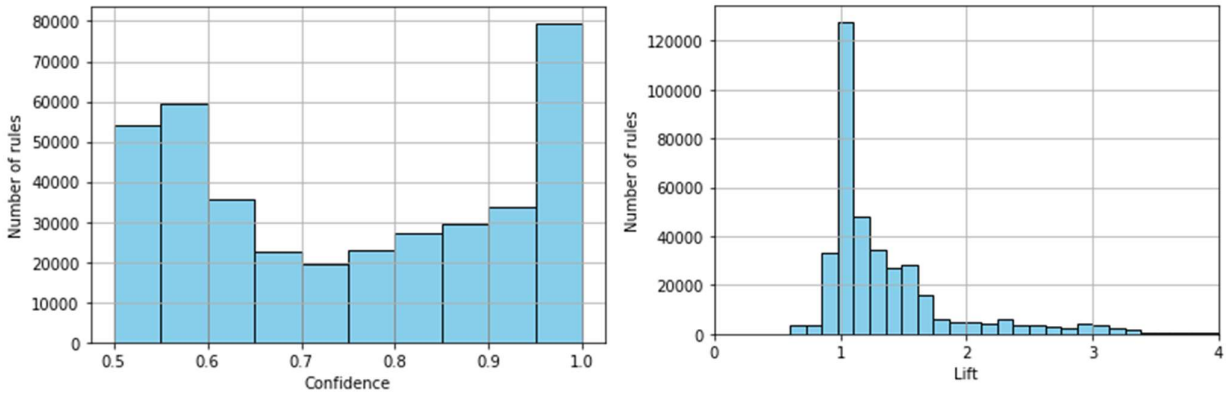
Focusing on the itemset with zmin equal to 9 the results are: 'MilesPerYear_in_[16919.167, 32944.333)', 'AcquisitionAveragePrice_in_[7389.5, 13510.0)', 'VehBCost_in_[6268.333, 12311.667)', 'CurrentAveragePrice_in_[6914.5, 13130.0)', 'WarrantyCost_in_[462.0, 1634.667)', 'AMERICAN', 'GOOD', 'AUTO', 'IsNotOnlineSale'.

b. Rules generation with different values of confidence

After searching all the frequent patterns with different parameters, the next step is about association rules extraction. The number of rules that can be extracted from frequent itemset depends on the value of confidence that is chosen. In the following graph is shown that the decreasing of the number of rules is correlated with the increasing of the confidence value from 50% to 98%.



We looked for the association rules with minimum support = 5% and zmin = 2 to analyse how the distribution of the rules changes with respect to the confidence. From the following histogram on the left we observe the distribution of the values of the confidence.



In particular this histogram shows that from the 384239 rules extracted the most of them has a value of confidence between 50% and 60% and between 90% and 100%, more precisely we found:
- 113736 rules with confidence between 50% and 60%;
- 58340 rules with confidence between 60% and 70%;
- 42319 rules with confidence between 700% and 80%;
- 56672 rules with confidence between 80% and 90%;
- 113172 rules with confidence between 90% and 100%;

c. Replacing missing values with the most meaningful rules

The association rules task is also useful for replacing missing values. The variable 'TopThreeAmericanName' presents four missing values which can be replaced by checking the rules implying high confidence with another variable taken as a reference point. Like at the beginning of the report, during the data understanding part, we use 'Make' as the reference variable for 'TopThreeAmericanName'. The starting point is to check the rules where the antecedent is equal to 'JEEP' in the variable 'Make' and the consequent is 'CHRYSLER' in the variable 'TopThreeAmericanName'.

- ('Make' = 'JEEP')⇨('TopThreeAmericanName' = 'CHRYSLER')
  support count = 1287, support = 2,20, confidence = 100%, lift = 3,12

This rule is a strong one because it has 100% confidence. In this case we can be sure that if a vehicle presents a missing value in the feature 'TopThreeAmericanName' and its 'Make' value is 'JEEP', with a 100% confidence we can replace this missing value with 'CHRYSLER'. The same concept is applied to replace all the remining missing values of 'TopThreeAmericanName' and 'Nationality'. All the rules can be seen below:

- ('Make' = 'DODGE')⇨ ('TopThreeAmericanName' = 'CHRYSLER')
  support count = 10355, support = 17,73, confidence = 100%, lift = 3.12
- ('Make' = 'GMC') ⇨('TopThreeAmericanName' = 'GM')
  support count = 502, support = 0.8 %, confidence = 100%, lift = 2,88
- ('TopThreeAmericanName' = 'GM') ⇨('Nationality' = 'AMERICAN')
  Support count = 20249, support = 34,68, confidence = 100%, lift = 1.19
- ('TopThreeAmericanName' = 'FORD') ⇨('Nationality' = 'AMERICAN')
  Support count = 9818, support = 16,81, confidence = 100%, lift = 1.19)

d. Rules generation to predict the target variable

The most interesting aspect that can return a lot of information to predict future bad purchases, is to study the rules that have as a consequent the target variable 'IsBadBuy'. With the parameters zmin = 2, support = 2% and confidence = 75% we extracted eleven rules that have 'IsBadBuy' equal to 1, as a consequent. Reaching a level of confidence equal to 75% in this situation is a good result because the classes of the target variable are very unbalance and the gap between the 12% ('IsBadBuy' = 1) and 75% is high. The three most interesting rules are the following:

- (‘NullWheelType’, 'VehBCost_in_[225.0, 6268.333)', 'AUTO', 'IsNotOnlineSale') ⇨('BAD')
  Support count = 928, Support= 1,59%, Confidence = 76,06%, Lift = 6.16
  With a good level of confidence (76,06%) can be said that if a vehicle presents an unknown type of wheel, its cost is between 225 and 6268, it has an automatic transmission and it wasn’t sold online than, it should be a bad purchase. From all the six ranges that were created for ‘VehBCost’, this is the lowest one and collects all the cheapest vehicles. The presence of 'AUTO' and 'IsNotOnlineSale' can be misleading because the classes are very unbalanced, but the most interesting features are the one linked with the kind of wheel type and the cost of the vehicle.
- (‘NullWheelType’, 'AcquisitionAveragePrice_in_[1269.0, 7389.5)', 'AUTO', 'IsNotOnlineSale')⇨('BAD')
  Support count = 936, Support = 1,60%, Confidence = 76,16 %, lift = 6.17
  This rule has the same antecedents as the first one, the only thing that changes is that ‘VehBCost’ is substituted with the value of the mean between all the average prices (that were 4) in range 1269 to 7389.5. But this situation reflects the first one because also in this case we have the lowest price range, so the cheapest vehicles.
- (‘NullWheelType’, 'MilesYear_in_[894.0, 16919.167)', 'AUTO', 'IsNotOnlineSale')⇨('BAD')
  Support count = 946, Support = 1,62%, Confidence = 75,32%, lift= 6.10
  In this last rule there isn’t information about the price, but we have a new variable that represents the mean of the kilometres made by a vehicle every year. The range is from 894 to 16919 and is the lowest one. This variable is the ratio between the total amount of the vehicle kilometres and its age, so this result can represent the older vehicles or the ones that were used less, and so with less kilometres. To check this condition we analyse the following table.

VehicleAge	1	2	3	4	5	6	7	8	9
MilesPerYearb									
MilesPerYear_in_[16919.167, 32944.333)	28	3155	11544	8882	2442	53	0	0	0
MilesPerYear_in_[32944.333, 48969.5)	1111	3509	26	0	0	0	0	0	0
MilesPerYear_in_[48969.5, 64994.667)	965	9	0	0	0	0	0	0	0
MilesPerYear_in_[64994.667, 81019.833)	270	0	0	0	0	0	0	0	0
MilesPerYear_in_[81019.833, 97141.151)	132	0	0	0	0	0	0	0	0
MilesPerYear_in_[894.0, 16919.167)	3	97	1133	4802	7861	6329	3724	1800	510

  We can see that all the oldest vehicles, with seven, eight and nine years belong to this range, and also the majority with five and six years. This result implies that is more likely that the older vehicles, even if with less kilometres, are bad purchases that the newer ones with more kilometres
  .

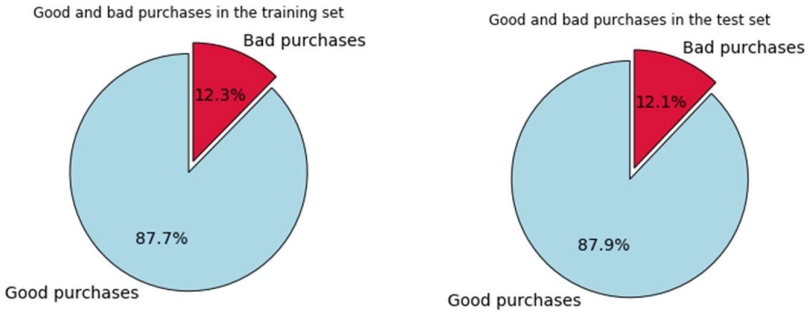
It’s very important to notice that there are three features that are present in all the antecedents that are: ‘NullWheelType’, 'IsNotOnlineSale' and 'AUTO'. The auction should be very careful with the vehicles that present this kind of characteristics and also with the cheapest and the oldest one.
We also checked the rules with IsBadBuy’ equal to 0 as a consequent. In this case the parameters can be higher: zmin equal to 2, support 40% and confidence 91%. The algorithm returns 17 rules, we analyse the most interesting one:

- ('Covers', 'AUTO', 'IsNotOnlineSale')⇨('GOOD')
  Support count = 22899, count = 39,22%, confidence = 92,15, lift = 1.05
  In this rule it’s important the presence of ‘Covers’ because it implies that the vehicle with this characteristic of the wheel are more likely to be a good purchase.
- ('AcquisitionAveragePrice_in_[7389.5, 13510.0)', 'VehBCost_in_[6268.333, 12311.667)', 'CurrentAveragePrice_in_[6914.5, 13130.0)', 'AUTO', 'IsNotOnlineSale') ⇨('GOOD')
  Support count = 23203, count = 39,74%, confidence = 91%, lift = 1.038
  Analyzing this rule is important to highlight all the price ranges. All these prices were divided in 6 ranges, all the ranges that are in this rule are the second, if we order them in increasing price order. This means that a very expensive vehicle isn’t necessarily a good purchase.

4. CLASSIFICATION

a. Variable preparation and dummy classifiers performance

For the classification task 80% of the values are for training and the other 20% are for the test set. The test set is already divided from the raining, so we continue by checking the composition of the test set and replacing missing values with the same method used on the training. Then we compared the distribution of the variable 'IsBadBuy', which is the predictive attribute, in the training and in the test. In the following graphs we can see that the training (51178 values are 0, 7208 are 1) and the test set (12829 values are 0, and 1768 are 1) are proportional.



Before starting, all the categorical attributes were transformed in a numerical format with the function 'OneHotEncoder' which assigns a value between 0 and the total number of labels minus one to every label.
At first we started the classification by analyzing the performances of the DummyClassifiers. This is a kind of classifier that uses simple rules, so it isn't very accurate, but its results are useful as baseline to compare the other more precise methods. In the following table there are the performances given by the DummyClassifiers with the 'stratified' strategy, where the predictions are generated by respecting the class distribution of the data and 'most frequent' strategy, where the classifier always predicts the most frequent class label in the data.

Stratified strategy

Performance on Training set

Accuracy 0.7836842375755915
F1-score [0.87669547 0.11951747]

	precision	recall	f1-score	support
0	0.88	0.88	0.88	51166
1	0.12	0.12	0.12	7207
accuracy			0.78	58373
macro avg	0.50	0.50	0.50	58373
weighted avg	0.78	0.78	0.78	58373

Performance on Test set

Accuracy 0.7848430861998081
F1-score [0.87751599 0.11599099]

	precision	recall	f1-score	support
0	0.88	0.88	0.88	12826
1	0.12	0.12	0.12	1768
accuracy			0.78	14594
macro avg	0.50	0.50	0.50	14594
weighted avg	0.79	0.78	0.79	14594

Most frequent strategy

Performance on Training set

Accuracy 0.8765353845099618
F1-score [0.93420608 0.00]

	precision	recall	f1-score	support
0	0.88	1.00	0.93	51166
1	0.00	0.00	0.00	7207
accuracy			0.88	58373
macro avg	0.44	0.50	0.47	58373
weighted avg	0.77	0.88	0.82	58373

Performance on Test set

Accuracy 0.8788543236946691
F1-score [0.93552152 0.00]

	precision	recall	f1-score	support
0	0.88	1.00	0.94	12826
1	0.00	0.00	0.00	1768
accuracy			0.88	14594
macro avg	0.44	0.50	0.47	14594
weighted avg	0.77	0.88	0.82	14594

Starting from this point only better results than those ones are acceptable.

b. Definition of different configuration of variables

To optimize the performance of the classification we tried four different configurations taking into account different composition of variables and every execution is evaluated with both kind of impurity measure Gini index and Entropy. For all the configurations we used Grid Searching to optimize two hyperparameters: min_sample_leaf which is the minimum number of samples required to be at a leaf node and min_sample_split

which is the minimum number of samples required to split an internal node. These two parameters are the ones that set the trade off between overfitting and underfitting. If they assume too low values, there will be overfitting problem, on the other hand if they are too high the issue will be underfitting. In the following point there are the results of the analysis:

- Configuration 1. For the first configuration are used all the new features defined before in data understanding ('MilesYear', 'Month', 'Day', 'AcquisitionAveragePrice', 'CurrentAveragePrice' and all the 12 differences between the prices). Then we also modify the variable 'WheelType' replacing its missing values with a specific feature 'NullWheelType' (we tried this step with all the variables with missing values but the results weren't significant). We create these features because they have a high impact on the implementation of the decision tree. In fact, in the following table 'Top five feature importance', there are the five variables that bring the higher contribution in achieving the classification task. All of the variables in the table are the new variable, so that means that they have an important in the implementation of the decision tree. In particular, a crucial role is played by 'WheelType' which percentage is 54%.
- Configuration 2. The only difference from the first configuration is that the missing values of the variable 'WheelType' are replaced with the mode between all the vehicles with the same manufacturer, model and submodel. Doing that the variable loses its importance in the contruction of the decision tree, that why it is missing in the table 'Top five feature importance'. This can be one of the reasons because the accuracy of the model decreases. The decrease of the accuracy from the first configuration can be seen in the table 'Configurations performance'.
- Configuration 3. In this configuration there are the same features like in the first one plus other variables. Regarding the prices we made the differences between all the combinations of the eight prices creating 28 new colums, plus the 8 that we already created from the difference between 'VehBCost' and the other eight prices. The other variables that are: 'WarrantyVehBCost' from the ratio between the cost of the warranty and 'VehBCost', 'VehBCostVehOdo' from the ration between 'VehBCost' and the odometer reading and 'VehBCostVehicleAge' from the ratio between 'VehBCost' and the age of the vehicle. In this configuration the feature 'WheelType' regain importance, even if its percentage is higher with the Gini Index, that with the Entropy. Some of the new variables are included in the five most important features. The performance of this configuration is better than the second one because the accuracy increases.
- Configuration 4. In this configuration we wanted to have a comparison to check if the new variables really improve the performance of the classification. In this classification we just add the variables 'MilesYear', 'Month', 'Day', 'AcquisitionAveragePrice', 'CurrentAveragePrice' and we keep the missing values of 'WheelType' with the feature 'NullWheelType'.

Top five feature importance			
Configuration 1	Gini index	WheelType	0.5429
		DiffCurrent_AcquisitionRetailAveragePrice	0.0666
		VehicleAge	0.0649
		DiffCurrent_AcquisitionAuctionAveragePrice	0.0474
		DiffCurrent_AcquisitionAuctionCleanPrice	0.0328
	Entropy	DiffCurrent_AcquisitionAuctionAveragePrice	0.1813
		VehicleAge	0.1333
		DiffCurrent_AcquisitionRetailAveragePrice	0.1023
		DiffCurrent_AcquisitionAuctionCleanPrice	0.0914
		Month	0.0637
Configuration 2	Gini index	DiffCurrent_AcquisitionAuctionAveragePrice	0.1870
		VehicleAge	0.1158
		DiffCurrent_AcquisitionAuctionCleanPrice	0.0918
		DiffCurrent_AcquisitionRetailAveragePrice	0.0914
		Month	0.0573
	Entropy	VehicleAge	0.1711
		DiffCurrent_AcquisitionAuctionAveragePrice	0.1022
		DiffCurrent_AcquisitionRetailAveragePrice	0.0861
		Month	0.07646
		DiffCurrent_AcquisitionAuctionCleanPrice	0.0736

Configuration 3	Gini index	WheelType VehBCostVehicleAge DiffAcquisition_CurrentRetailAveragePrice DiffAcquisition_CurrentAuctionAveragePrice DiffAcquisition_CurrentAuctionCleanPrice	0.5350 0.09718 0.0595 0.0389 0.0226
	Entropy	WheelType VehicleAge VehBCostVehicleAge VNZIP1 DiffAcquisition_CurrentAuctionAveragePrice	0.2837 0.0578 0.0459 0.0310 0.0287
Configuration 4	Gini index	WheelType Auction VehicleAge VNZIP1 VehBCost	0.5081 0.0690 0.0631 0.04542 0.04188
	Entropy	WheelType VehicleAge Auction VehBCost VNZIP1	0.4155 0.1110 0.0545 0.0535 0.0428

Configurations performances									
		Configuration 1		**Configuration 2**		**Configuration 3**		**Configuration 4**	
		Gini index	Entropy	Gini index	Entropy	Gini index	Entropy	Gini index	Entropy
Accuracy	training	0,9007	0,9055	0,8856	0,8825	0,901	0,9057	0,9016	0,8999
	test	0,9008	0,8941	0,8996	0,8989	0,9002	0,8941	0,898	0,8974
Precision	training	0,9	0,9	0,87	0,86	0,9	0,9	0,89	0,9
	test	0,9	0,88	0,84	0,85	0,9	0,88	0,89	0,89
Recall	training	0,9	0,91	0,89	0,88	0,9	0,91	0,9	0,9
	test	0,9	0,89	0,9	0,9	0,9	0,89	0,9	0,9
f1-score	training	0,87	0,89	0,85	0,84	0,88	0,88	0,88	0,87
	test	0,87	0,87	0,86	0,86	0,87	0,87	0,87	0,87
Precision	training[0]	0,9	0,91	0,89	0,89	0,9	0,91	0,9	0,9
	test[0]	0,9	0,9	0,9	0,91	0,9	0,9	0,9	0,9
Recall	training[0]	1	0,99	0,99	0,99	0,99	0,99	0,99	1
	test[0]	1	0,98	0,99	0,99	1	0,98	0,99	0,99
f1-score	training[0]	0,95	0,95	0,94	0,94	0,95	0,95	0,95	0,95
	test[0]	0,95	0,94	0,95	0,95	0,95	0,94	0,94	0,94
Precision	training[1]	0,87	0,82	0,74	0,64	0,85	0,82	0,82	0,87
	test[1]	0,88	0,67	0,27	0,33	0,86	0,68	0,76	0,79
Recall	training[1]	**0,23**	**0,3**	**0,11**	**0,11**	**0,24**	**0,3**	**0,26**	**0,22**
	test[1]	**0,21**	**0,25**	**0,02**	**0,04**	**0,21**	**0,24**	**0,23**	**0,21**
f1-score	training[1]	0,36	0,44	0,2	0,19	0,37	0,44	0,39	0,36
	test[1]	0,34	0,37	0,04	0,07	0,34	0,35	0,35	0,33
Min sample split		170	80	10	130	170	80	110	15

Min Sample leaf		70	15	80	60	70	20	35	70

In the last table there are the index to evaluate the performance of the classification. The higher accuracy in both test and training is given by the first and the third configuration, both with the Gini index and the worst results are given by the second configuration. Nevertheless, the level of accuracy is always around 90%. To have a better understanding of the goodness of these models we also insert in the table the measure: precision, recall and F1-score and we calculate them also with the respect of both the class 1 and 0 of 'IsBadBuy'. The values of precision are high, but the values of the recall is very low. This problem is due to the fact that the two class 0 and 1 are very unbalanced like we can see from the pie charts at the beginning, that's why the recall is lower. We tried to solve this problem with two methods: by doing oversampling and adding the cost matrix. The first method didn't give us great result, so we decide to use the cost matrix. We gave more weight at the class 'IsBadBuy' = 1, which is the smaller class, trying to balance the data. The two following tables have the same purposes as the two before, but the data are update with the new weighted classes. In the first row of the second table there are the weight given to every class, and we can see that the recall of the class 'IsBadBuy' = 1 is improved at the expense of the precision.

Top five feature importance with the class weights			
Configuration 1	Gini index	WheelType	0,473
		VehicleAge	0,133
		VehBCost	0,053
		DiffCurrent_AcquisitionRetailAveragePrice	0,034
		DiffCurrent_AcquisitionAuctionCleanPrice	0,028
	Entropy	WheelType	0,394
		VehicleAge	0,136
		VehBCost	0,044
		DiffCurrent_AcquisitionAuctionCleanPrice	0,037
		DiffCurrent_AcquisitionAuctionAveragePrice	0,033
Configuration 2	Gini index	VehicleAge	0,178
		DiffCurrent_AcquisitionAuctionAveragePrice	0,140
		DiffCurrent_AcquisitionRetailAveragePrice	0,085
		VehBCost	0,069
		DiffCurrent_AcquisitionAuctionCleanPrice	0,069
	Entropy	VehicleAge	0,156
		DiffCurrent_AcquisitionAuctionAveragePrice	0,097
		DiffCurrent_AcquisitonRetailCleanPrice	0,078
		DiffCurrent_AcquisitionAuctionCleanPrice	0,071
		DiffCurrent_AcquisitionRetailAveragePrice	0,065
Configuration 3	Gini index	WheelType	0,463
		VehBCostVehicleAge	0,164
		DiffAcquisition_CurrentRetailAveragePrice	0,025
		VehOdo	0,018
		DiffAcquisition_CurrentAuctionAveragePrice	0,016
	Entropy	WheelType	0,384
		VehicleAge	0,102
		VehBCostVehicleAge	0,064
		DiffAcquisition_CurrentAuctionAveragePrice	0,030
		DiffAcquisition_CurrentRetailAveragePrice	0,024
Configuration 4	Gini index	WheelType	0,494
		VehicleAge	0,139
		VehBCost	0,062
		VNZIP1	0,026
		VehOdo	0,024

	Entropy	WheelType VehicleAge VehBCost Auction VNZIP1	0,415 0,141 0,061 0,037 0,034

Configurations performances with the class weight									
		Configuratio 1		**Configuration 2**		**Configuration 3**		**Configuration 4**	
		Gini index	Entropy	Gini index	Entropy	*Gini index*	Entropy	Gini index	Entropy
Class weight		0: 1 1: 3.4	0: 1 1: 3.4	0: 1 1: 2.8	0: 1 1: 3.0	*0: 1 1: 3.4*	0: 1 1: 3.4	0: 1 1: 3.4	0: 1 1: 3.4
Accuracy	training	0,858	0,86	0,854	0,851	*0,863*	0,861	0,865	0,861
	test	0,837	0,839	0,843	0,839	*0,842*	0,843	0,847	0,836
Precision	training	0,86	0,86	0,85	0,85	*0,87*	0,87	0,86	0,86
	test	0,84	0,84	0,84	0,84	*0,85*	0,85	0,84	0,84
Recall	training	0,86	0,86	0,85	0,85	*0,86*	0,86	0,86	0,86
	test	0,84	0,84	0,84	0,84	*0,84*	0,84	0,85	0,84
f1-score	training	0,86	0,86	0,85	0,85	*0,87*	0,86	0,86	0,86
	test	0,84	0,84	0,84	0,84	*0,84*	0,84	0,85	0,84
Precision	training[1]	0,43	0,44	0,4	0,39	*0,45*	0,44	0,45	0,44
	test[1]	0,34	0,35	0,19	0,2	*0,36*	0,36	0,36	0,34
Recall	training[1]	0,48	0,47	0,36	0,38	*0,48*	0,48	0,45	0,46
	test[1]	0,38	0,37	0,19	0,21	*0,37*	0,37	0,35	0,36
f1-score	training[1]	0,46	0,45	0,38	0,39	*0,46*	0,46	0,45	0,45
	test[1]	0,36	0,36	0,19	0,2	*0,36*	0,36	0,36	0,35
Min Sample split		15	210	210	10	*210*	210	210	10
Min sample leaf		100	100	100	100	*100*	100	100	100

c. Best decision tree

We think that the best configuration is the third one with the Gini index and the weight for the target variable, here there is its confusion matrix:

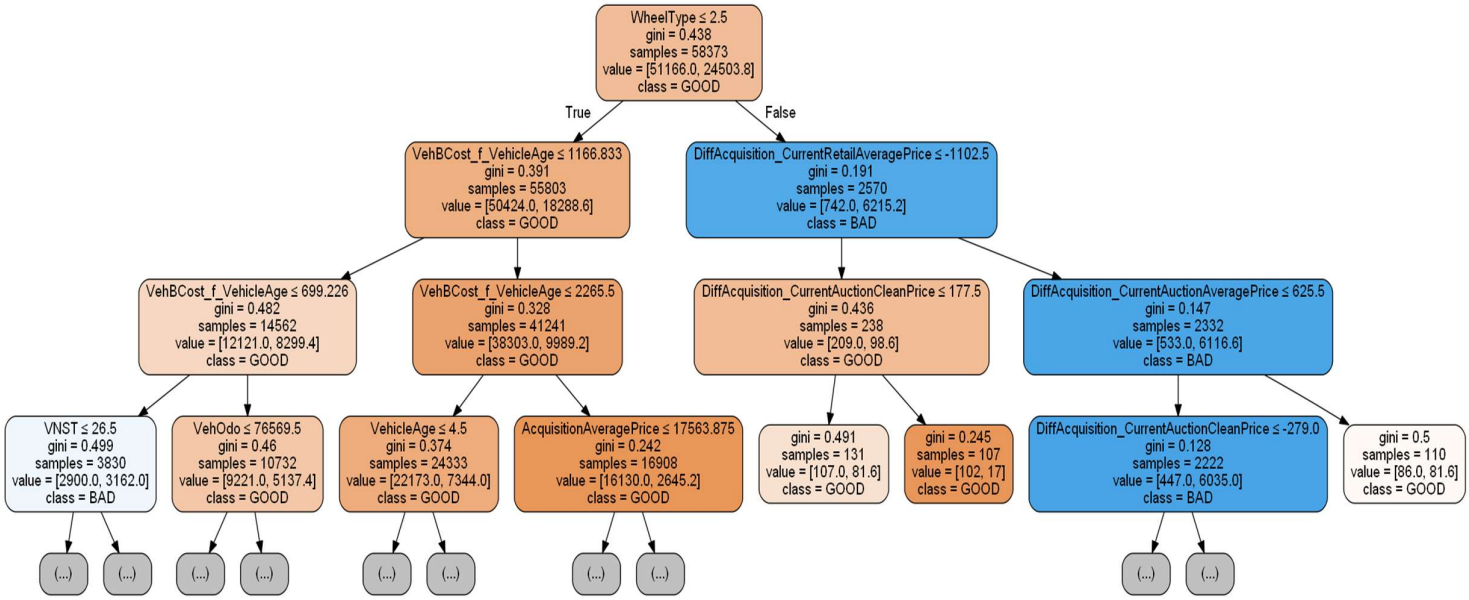
	Predicted positive	Predicted negative
Actual positive	11632	1197
Actual negative	1109	659

Accuracy 0.8420223333356169
F1-score [0.90981619 0.36368653]

	precision	recall	f1-score	support
0	0.91	0.91	0.91	12829
1	0.36	0.37	0.36	1768
accuracy			0.84	14597
macro avg	0.63	0.64	0.64	14597
weighted avg	0.85	0.84	0.84	14597

The min_sample_leaf and min_sample_split parameter were choose respectively equal to 100 and 210 to avoid overfitting or underfitting, in fact the performance in the training and in the test are very similar. We choose this one because the class of the target variable is very unbalanced (12,3% bad purchases and 87,7%

good ones), so in this case the accuracy isn't very reliable as valuation metric. To be able to predict the bad purchases we have to increase the recall value and to do so we must increase the true positive and at the same time decrease the false negative in fact the recall is the ration between true positive and the sum of true positive with false negative. That is the reason why we prefer the configuration with the weight, which allow us to have a higher recall value, even if the accuracy is lower. Another consequence of this choice is that also the level of precision decreases a little, this means that it's more likely that the company decides to not buy a vehicle that can be a good purchase and a potential income for the auction.
In the following image can be seen the starting point of the decision tree with the parent root and the first four level of the tree. We can see that these first split are made using the variables that are presented in the table containing the most important features for the classification.



On the Kaggle competition we decided to upload the third configuration with the Gini index but without the weight for the classes because the purpose was to maximize the accuracy because the results were calculated with F1-score.

CONCLUSION

From all the results given by the previous algorithms we want to highlight the information that can be useful for the auction. One of the most interesting and unexpected things is that the missing value of the variable 'WheelType' are very important to forecast if a future purchase is a bad or a good one. The association rules and the classification show how strong is the link between the bad purchases and the missing values in this variable to such an extent that this kind of vehicle are considered unreliable deal. Another important aspect is that the auction has to be very careful with the cheapest and the oldest vehicles. The most reliable are the ones in the middle range of price and the newer cars, even if they have more kilometers than the older ones.

Additional Task (DM 9 CFU)

Marco Mazzei (546816)

Classification with K-Nearest Neighbors

In this final part we compare the results of classification by decision tree with K-Nearest Neighbors (K-NN). As the target class is unbalanced (87,5-12,5%) we don't expect good results in predicting the bad purchases from this classificator without an oversampling of the minority class (or undersampling of the majority one). Training and test set are already splitted (as shown in the DT's chapter), so we basically did the same steps of the DT's chapter to prepare the data for this classifier (replaced missing values on both training and test with the same methods and one hot encoded).
Initially we evaluated K-NN's performances without oversampling the minority class, and we observed how the results changed with different values of the following parameters:

- k (number of nearest neighbours), we tested from k=5 to k=500 with steps of 5 (so k=5,10,15,…,500). Of course this is the most important parameter of K-NN, it regulates the trade-off between overfitting and underfitting (k too low ,for example k=1, overfits the training set, while k too high ,for example k=n.training records, underfits the data)
- metric, the metric for the distance function, we used 'minkowski' that with p=2 is equivalent to the standard Euclidean metric
- p, is the power parameter for the Minkowski metric (as said before p=2 is Euclidean distance, p=1 is Manhattan distance). We tested p=2,3,4.
- weights:

  -'uniform': uniform weights. All points in each neighbourhood are weighted equally.
  - 'distance': weight points by the inverse of their distance. In this case, closer neighbours of a query point will have a greater influence than neighbours which are further away.

In order to try this combination of parameters maximizing the overall accuracy we used Grid Search.
The following table recap some of the most interesting configurations of parameters found:

K	**45**	**80**	**35**	**70**
weights	uniform	uniform	distance	distance
minkowski_p	2	2	2	2
Accuracy_training	0.877	0.876	1	1
Accuracy_test	0.879	0.879	0.879	0.879
Precision_training	0.86	0.77	1	1
Precision_test	0.84	0.77	0.83	0.89
Recall_training	0.88	0.88	1	1
Recall_test	0.88	0.88	0.88	0.88
f1-score_training	0.83	0.82	1	1
f1-score_test	0.83	0.82	0.82	0.82
Precision_training[0]	0.88	0.88	1	1
Precision_test[0]	0.88	0.88	0.88	0.88
Recall_training[0]	1	1	1	1
Recall_test[0]	1	1	1	1
f1-score_training[0]	0.93	0.93	1	1
f1-score_test[0]	0.94	0.94	0.94	0.94
Precision_training[1]	0.70	0.00	1	1
Precision_test[1]	0.57	0.00	0.47	1

Recall_training[1]	0.00	0.00	1	1
Recall_test[1]	0.00	0.00	0.01	0.00
f1-score_training[1]	0.01	0.00	1	1
f1-score_test[1]	0.00	0.00	0.01	0.00

As expected, also changing almost all the parameters, this classifier without an oversampling of the minority class (IsBadBuy=1) is not able to spot well the bad purchases.
When as the parameter weights we used 'distance' K-NN overfits the training data (100% accuracy) but has almost the same results of the previous ones on the test.

As discussed before, as a partial solution for the bad performance on predicting the minority class, we did an oversampling of this class. In particular we did a random oversampling of the rows with IsBadBuy=1 and the new training set had a distribution bad-good 35-75%. We decided this percentages after some tests, obviously in order to have better performances on bad purchases we could have increased more the "bad" percentage, but this would have led to a drop in overall accuracy.
The following table recap the best configuration found with the oversampled data in order to predict better the bad purchases without losing much overall accuracy:

K	**45**	**70**	**35**	**60**
weights	uniform	uniform	distance	distance
minkowski_p	2	2	2	2
Accuracy_training	0.713	0.711	1	1
Accuracy_test	0.792	0.811	0.783	0.802
Precision_training	0.70	0.73	1	1
Precision_test	0.81	0.82	0.81	0.90
Recall_training	0.71	0.90	1	1
Recall_test	0.79	0.81	0.78	0.88
f1-score_training	0.69	0.81	1	1
f1-score_test	0.80	0.81	0.80	0.89
Precision_training[0]	0.74	0.73	1	1
Precision_test[0]	0.90	0.90	0.90	0.23
Recall_training[0]	0.87	0.90	1	1
Recall_test[0]	0.86	0.89	0.85	0.26
f1-score_training[0]	0.80	0.81	1	1
f1-score_test[0]	0.88	0.89	0.87	0.24
Precision_training[1]	0.60	0.61	1	1
Precision_test[1]	0.22	0.23	0.21	0.23
Recall_training[1]	0.39	0.33	1	1
Recall_test[1]	0.28	0.25	0.29	0.26
f1-score_training[1]	0.47	0.42	1	1
f1-score_test[1]	0.24	0.24	0.24	0.24

Conclusion

With the decision tree classifier without any doubt we obtained better result when we focused on overall accuracy.
Also when the focus was on the bad purchases (IsBadBuy=1) the DT with the weights had better performances on the test set than K-NN with oversampling.
In terms of computational time we observed that, as expected from the theory, K-NN is slower than the DT on predicting the label of an unseen record, while for the construction of the classifiers they are both fast.